Constraint Based World Modeling for Multi Agent Systems in Dynamic Environments

D I S S E R T A T I O N

zur Erlangung des akademischen Grades

Dr. rer. nat.
im Fach Informatik

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät II
Humboldt-Universität zu Berlin

von
Dipl.-Inf. Daniel Göhring
06. Juni 1978

Präsident der Humboldt-Universität zu Berlin:
Prof. Dr. Christoph Markschies

Dekan der Mathematisch-Naturwissenschaftlichen Fakultät II:
Prof. Dr. Peter Frensch

Gutachter:
1. Prof. Dr. Hans-Dieter Burkhard
2. Prof. Dr. Uwe Schwiegelshohn
3. Prof. Rick Middleton

eingereicht am: 14. April 2009
Tag der mündlichen Prüfung: 10. November 2009

For my parents.

Abstract

Mobile autonomous robotics is a very young and complex field of research. Only in recent decades have robots become able to explore, to move, navigate and to interact with their environment.

Since the world is uncertain and since robots can only gain partial information about it, probabilistic navigation algorithms have become very popular whenever a robot has to localize itself or surrounding objects. Furthermore, cooperative exploration and localization approaches have become very relevant lately, as robots begin to act not just alone but in groups. Within this thesis a new approach using the concept of spatial percept-relations for cooperative environment modeling is presented and evaluated. As a second contribution, constraint based localization techniques will be introduced for having a robot or a group of robots efficiently localized and to model their environment.

Zusammenfassung

Die mobile Robotik stellt ein sehr junges und komplexes Forschungsfelder unserer Zeit dar. Innerhalb der letzten Jahrzehnte wurde es Robotern möglich, sich innerhalb ihrer Umgebung zu bewegen, zu navigieren und mit ihrer Umwelt zu interagieren.

Aufgrund der Tatsache, dass die Welt von Unsicherheit geprägt ist und ein Roboter immer nur partielle Information über sie erhalten kann, wurden probabilistische Navigationsverfahren entwickelt, mit denen sich Roboter lokalisieren und Objekte ihrer Umgebung modellieren können. Weiterhin wurden in letzter Zeit Verfahren untersucht, die die kooperative Exploration der Umgebung durch eine Gruppe von Robotern zum Ziel haben. In der vorliegenden Arbeit wird ein neuartiges Konzept, welches sich Perzeptrelationen für die kooperative Umweltmodellierung zu Nutze macht, vorgestellt und evaluiert. Einen zweiten Beitrag der Arbeit stellen constraintbasierte Lokalisierungstechniken dar, die es einem oder mehreren Robotern auf effiziente Art und Weise ermöglichen, sich zu lokalisieren und ihre Umwelt zu modellieren.

Contents

Preface

This dissertation thesis describes the work of four years of research. During my work on different fields within mobile robotics it became clear to me that modeling the world in which the robot has to act and to interact with other robots is a highly interesting research topic. The question of modeling the environment is even more interesting for legged robots as four-legged or biped robots, which have to handle high motion and sensory noise and the research within this field has just begun recently.

Language and Annotations

Robotics is a very interdisciplinary and technologically advanced field of research. The main language for communication here is English. Translating some important concepts can sound uncommon to readers who only know the English terms. To give access to this work to as many people as possible, it will be written and published in English.

Annotation of mathematic formulas and symbols was to be geared to the works of Thrun, Burgard and Fox [114]. Introduced mathematical symbols will be used consistently over different chapters.

Thanks

First and foremost I would like to thank my adviser Prof. Dr. Hans-Dieter Burkhard. My interest in robotics was encouraged by him as he provided the knowledge and guidance necessary to the successful completion of this thesis. Working with him on the RoboCup team and sharing so many crunch-time moments was very memorable, most important, I learned a lot from these treasured experiences.

I also want to extend my gratitude to Prof. Dr. Uwe Schwiegelshohn for the joyful discussions and hints to make this work clearer to understand. I have had the pleasure of working with Prof. Rick Middleton at the Hamilton Institution in Ireland, whose creativity and great ideas taught me to see things from a different perspective and helped me to drive things forward.

I want to thank my colleagues at the Humboldt-Universität zu Berlin for the inspiring discussions, Dr. Manfred Hild for his refreshing ideas, Uwe Düffert, who together with Matthias Jüngel helped me to understand basic concepts, especially when I started my research in the Aiboteam. I want to thank Jan Hoffmann who supervised my undergrad works. I thank Heinrich Mellmann for the joyful work on the Nao-robot and the many research nights at the institute where we had to reach some deadlines. Furthermore I would like to say thanks for the fruitful discussions to Benjamin Altmeyer, Ralf

Berger, Kateryna Gerasymova, Viviana Götzke, Martin Lötzsch, Torsten Siedel, Michael Spranger, Tim Laue, Max Risler, and Michael Quinlan.

For important hints during the final phase of this work I would like to say thanks to Maj-Britt Isberner and to Christine Zarges. Their inspirations have helped to increase the quality of this work. Most important I want to thank my family for giving me so much moral support.

1 Introduction

Building intelligent machines is a century old human dream. One purpose of having intelligent robots is to let them do repetitive, uninteresting, hard or dangerous work, or work that requires high accuracy, attention and speed. Initially, many robots or robot manipulators were immobile. They needed to operate in very controlled environments with clearly defined tasks and therefore, were able to act within the sensing and computing limits of the time. Improving technology allowed the controllers of robots to become more complex, and so the tasks for robots became more complex as well. Especially the continued development of electronics, integrated circuitry and software played an important role for the abilities of robots. Many of those machines are described as "intelligent", even though there is no clear definition about what intelligence really means.

With progress in mechanical design, robots became mobile. There are many different techniques available to have a robot moving, e.g., wheel driven, legged, flying, swimming and many more possibilities. With the ability to move autonomously the question of navigation becomes important. The task of navigation within dynamic environments [34] has been summarized in [75] to the questions: *Where am I?, Where am I going? How should I get there?* It becomes more challenging, when other objects have to be tracked, e.g., an autonomous car should be aware of the position, motion direction and speed of other cars. As a second example, an autonomous lawnmower has to stay away from humans and to be able to navigate within a garden with as few pre-calibration necessary as possible. Finding its way back to the docking station would be appreciated for such a robot as well. Traction problems on uneven ground complicate the navigation as well as dead reckoning may become very unreliable. Those machines are already available, even if usually just partially fulfilling the here presented requirements.

1.1 Perception, Uncertainty and Robustness

Perception is the process by which sensory inputs are selected, organized and interpreted into the agent's knowledge of the world [91]. It is the basis for all decisions the agent has to take. In [69] Kwok gave an example of human perception, where the visual sensors are the eyeballs and the visual input are the images we see. Upon receipt of those images, the brain makes "mental pictures" and integrates their information into our internal knowledge, which allows us to interact with these objects. This process can be described as *passive perception*. Another possibility of perception is to keep track of certain parts of the environment by moving the eyeballs. This sensor control to gather more information based on our knowledge is the *active* side of perception.

One aspect of robot perception is *partial observability*. Robots have sensors that are limited in their range and coverage. Furthermore, robots can only perceive a part of their surroundings, leading to partial information about the world, especially when the world is highly dynamic.

Sensors have intrinsic errors. Even laser range finders, providing millimeter accuracy, have small errors and, as stated in [69]: "... their accumulation over time will snowball into magnitudes that cannot be ignored". This incompleteness and inaccuracy of sensory information leads to *uncertainty* within the agents belief of the world. Modeling algorithms used on a robot have to be *robust* to those errors. Robustness means that a robot is able to filter out noisy sensory data to make its own world model more accurate. Moreover, localization techniques have to work under real-time constraints, and thus have to be computationally efficient.

1.2 Multi-Agent Localization

A lot of work has been done so far on single-robot localization and tracking. Examples are the Minerva tour guide robot [8] or robots within the RoboCup domain [73, 97]. Nowadays, robots often have to work within a group, e.g., in [11] a group of agents has to explore an unknown area collaboratively. Thus, an important field of research is perception, localization and planning of a group of robots to solve a task together. For multi-agent localization and world modeling the following questions arise:

- How can information be interchanged?
- How can the environment cooperatively be explored?
- Can the task be split up in subtasks and how?
- Shall a centralized approach be used or a decentralized one?

Cooperative world modeling, for example, can have a benefit for all participating robots because of the limited perception abilities of each agent. For a single agent it is usually impossible to keep track of its whole surroundings. Camera images have a limited field of view combined with a limited resolution and contrast. Or there can be parts of the environment not directly observable from an agents position, because they are occluded by other objects. Splitting up the perception task into subtasks, such that each robot observes a different area and communicates its knowledge to the other robots, allows each agent to create a model of the whole scene.

This thesis will focus on the interchangeability of perception data within a group of agents, under the assumption that interchanging sensory data will improve, or at the very least, should not worsen the model of the environment for each robot. The integration of the sensory data into a model has to work under real-time conditions, so the underlying algorithms have to be very efficient. Therefore, within this thesis constraint based localization methods will be introduced and analyzed with regard to their applicability to the modeling task.

1.3 Thesis Statement

Modeling of the environment by multiple agents and under real-time constraints will be improved when robots communicate and cooperate, i.e. interchange information about their surroundings. Constraint Based modeling techniques enable robots to maintain a computationally efficient model of their environment.

1.4 Thesis Contributions

The contributions within this work are mainly related to perception and modeling. They can be summarized as follows:

- It will be discussed, which properties of sensory data are appropriate for communication. Therefore the concept of using spatial relations between objects will be introduced.
- An approach to modeling dynamics by a group of agents using percept-relations will be introduced.
- Communication principles and its effects on modeling accuracy are discussed.
- An approach to handling communication delays within the modeling process is introduced.
- A constraint based modeling technique is developed, which enables a robot to efficiently represent its belief about the environment (including self-localization and object tracking).
- Different strategies to find and to handle inconsistent, i.e., erroneous sensory data are introduced.
- Constraint based modeling approaches are compared to other modeling techniques.
- A discussion is given on how multi-modal egocentric distributions can be transformed into allocentric distributions, to be communicated to other agents.
- Within this thesis will be presented, how constraint based modeling approaches can be applied to multi-agent multi-object tracking. Therefore, constraint based data association techniques for tracking multiple objects by a group of robots are introduced.

The work is structured as follows: In Chapter 2 an introduction into the basic concepts of theory of probability is given and basic modeling terms are presented. Those concepts are used to describe the most widely used Bayesian filtering algorithms in Chapter 3. Chapter 4 introduces the concept of percept-relations as a possibility to interchange sensory data between different robots and as a form of input data for object

tracking and self-localization. In Chapter 5 is presented how spatial percept-relations can be used for object velocity estimation by a group of robots. Possibilities of handling delayed communicated sensory data are discussed. The theoretical background for constraint based localization approaches is introduced in Chapter 6. In Chapter 7, an implementation for a constraint based self-localization is introduced. Its properties are analyzed and compared to other widely used modeling techniques. Chapter 8 focusses on constraint based multi-agent object tracking. Algorithms as constraint based model transformation and data association are described. The work is summarized in Chapter 9.

2 Preliminaries

This chapter will introduce important concepts that are necessary for object modeling. Many of those concepts refer to probability theory, which has become very important for handling uncertain sensory data.

2.1 Basics of Probability Theory

In this section basic terms of probability theory are described. Variables, that are part of a state vector are called state variables. Let X be a probabilistic variable and x a value, that X can take. Then x is called an *elementary event*. The probability of X to take value x is denoted by p:

$$p(X = x) \tag{2.1}$$

Following Kolmogorov, we assume that probabilities can only be non-negative:

$$p(X = x) \geq 0 \tag{2.2}$$

For discrete elementary events, the sum of the probabilities of all elementary events x_i, is 1:

$$\sum_X p(X = x) = 1 \tag{2.3}$$

In many applications including mobile robotics some state variables are continuous and can take infinite values. Therefore, the state space is called *continuous*, e.g., as for position variables on a plane. The probability of a continuous variable to take a certain value is described by a *probability density function (abbreviated by: pdf)*. In this work, it is assumed that all continuous random variables are measurable and that all continuous distributions possess densities [114]. A pdf over a continuous variable always integrates to 1:

$$\int p(x)\mathrm{d}x = 1 \tag{2.4}$$

Continuous pdfs are not upper bound, as their values describe probability densities and not probabilities.
The expectation $E_D(X)$ or μ of a probability variable X can be calculated for discrete elementary events as:

$$E_D[X] = \sum_i x_i p(X = x_i) \tag{2.5}$$

For continuous states and for a given density function f the expectation $E_C[X]$ can be

calculated as:

$$E_C[X] = \int_{-\infty}^{\infty} x f(x) \mathrm{d}x \tag{2.6}$$

The n-th moment is defined for discrete state variables as:

$$E_D[X^n] = \sum_i x_i^n p(X = x_i) \tag{2.7}$$

or, for the continuous case respectively as:

$$E_C[X^n] = \int_{-\infty}^{\infty} x^n f(x) \mathrm{d}x \tag{2.8}$$

Besides those defined moments, *central moments* consider the distribution around the expected value. The second central moment, usually called variance, can be calculated for discrete state variables as:

$$\mathrm{var}(X) \triangleq E_D[(X - E_D[X])^2] = \sum_i (x_i - E_D[X])^2 p(X = x_i) \tag{2.9}$$

For continuous probabilistic variables the variance can be calculated as:

$$\mathrm{var}(X) \triangleq E_C[(X - E_C[X])^2] = \int_{-\infty}^{\infty} (x - E_C[X])^2 f(x) \mathrm{d}x \tag{2.10}$$

The square-root of the variance is called *standard deviation* and is commonly denoted by σ. The covariance of two probabilistic variables X and Y with expectation $E_D[X]$ and $E_D[Y]$ can be defined for discrete probabilistic variables as:

$$\begin{aligned} \mathrm{cov}(X, Y) &\triangleq E_D[(X - E_D(X))(Y - E_D(Y))] && (2.11)\\ &= \sum_i \sum_j (x_i - E_D[X])(y_j - E_D[Y]) p(x_i, y_j) && (2.12)\\ &= \sigma_{X,Y}^2 && (2.13) \end{aligned}$$

and for continuous variables as:

$$\begin{aligned} \mathrm{cov}(X, Y) &\triangleq E_C[(X - E_C(X))(Y - E_C(Y))] && (2.14)\\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x - E_C[X])(y - E_C[Y]) f(x, y) \mathrm{d}x \mathrm{d}y && (2.15)\\ &= \sigma_{X,Y}^2 && (2.16) \end{aligned}$$

The correlation coefficient ρ of two probabilistic variables X and Y is calculated as:

$$\rho_{XY} \triangleq \frac{\sigma_{XY}^2}{\sigma_X \sigma_Y}, \tag{2.17}$$

where σ_X and σ_Y are the standard deviations of X and Y. In case of multi-dimensional

state spaces the covariance matrix contains the covariances for all pairs of probabilistic variables of the state space. The covariance matrix is positive semi-definite and symmetric. The variances of the state variables lie on the main axis of the covariance matrix. The *joint probability* of two independent random variables X and Y is calculated as:

$$p(x,y) = p(x)p(y) \tag{2.18}$$

The theorem of total probability states that the probability of event x can be derived from the common probabilities of x with all possible values y_i of Y:

$$p(x) = \sum_i p(x,y_i), \tag{2.19}$$

or in the continuous case, as an integral:

$$p(x) = \int p(x,y)\mathrm{d}y \tag{2.20}$$

There are different ways to write Bayes law. Within Bayes law the conditional probability $p(x|y)$ is set in relation to $p(y|x)$. For discrete probabilities Bayes law can be written as:

$$p(x|y) = \frac{p(x,y)}{p(y)} = \frac{p(x,y)}{\sum_i p(x_i,y)}, \tag{2.21}$$

and for the continuous case as:

$$p(x|y) = \frac{p(x,y)}{p(y)} = \frac{p(x,y)}{\int p(x,y)\mathrm{d}x} \tag{2.22}$$

Since $p(y)$ is constant for the calculation of $p(x|y)$, given different values of x, $p(y)^{-1}$ is often denoted by normalizing constant η, to have all probabilities summing up to 1:

$$p(x|y) = \eta p(y|x)p(x) \tag{2.23}$$

Furthermore, with equation (2.21) and (2.22) one can derive the joint probability of two not necessarily independent variables:

$$p(x,y) = p(x|y)p(y) \tag{2.24}$$

The probability of an event x can depend on many other variables. So, for example, when $X = x$ depends on two variables Y and Z we get:

$$p(x|y,z) = \frac{p(y|x,z)p(x|z)}{p(y|z)}, \tag{2.25}$$

for $p(y|z) > 0$.

Two variables (X and Y) are said to be conditionally independent, when the condi-

tional dependencies can be decomposed as follows:

$$p(x, y|z) = p(x|z)p(y|z) \tag{2.26}$$

Equivalently, X and Y are conditionally independent, if the following equations hold:

$$p(x|z) = p(x|y, z) \tag{2.27}$$
$$p(y|z) = p(y|x, z) \tag{2.28}$$

Conditional independencies play an important role in robotics for object modeling. With this concept probability calculations can often be simplified, by just taking probabilities into account that directly affect others.

Another important modeling concept is entropy. This concept from information theory can give hint about the distribution, e.g., if the probability density function is constant over the whole state space or if some events are more likely than others. The entropy $H_p(x)$ over a distribution $p(x)$ is calculated as:

$$H_p(x) = E[-\log_2 p(x)], \tag{2.29}$$

which leads for discrete probabilistic variables to

$$H_p(x) = \sum -p(x) \log_2 p(x) \tag{2.30}$$

and for continuous ones to:

$$H_p(x) = \int -p(x) \log_2 p(x) \mathrm{d}x \tag{2.31}$$

Note that in (2.30), (2.31) we adopt the convention that $0 \cdot \log(0) \triangleq 0$.

2.2 Normal Distribution

Normal or Gaussian distributions are one of the most widely used form of probabilistic density functions, especially within the object modeling field. After the Lindeberg-Levy central limit theorem (CLT), the normalized sum of a large number of independent and equally distributed probabilistic variables will be approximately standard normally distributed [96].

Normal distributions are used in mobile robotics to estimate the positions of different objects under certain assumptions. One assumptions, e.g., for Kalman filters is that sensory and process noise are normally distributed. Usually normal distributions represent the position probability of a single object within a domain of positions. Therefore, normal distributions belong to the class of unimodal distributions. In many other cases the robot cannot infer its exact position, e.g., when sensory data is ambiguous. In those cases multiple possible positions remain. The probability density function of the robot positions is multimodal and cannot be described accurately by

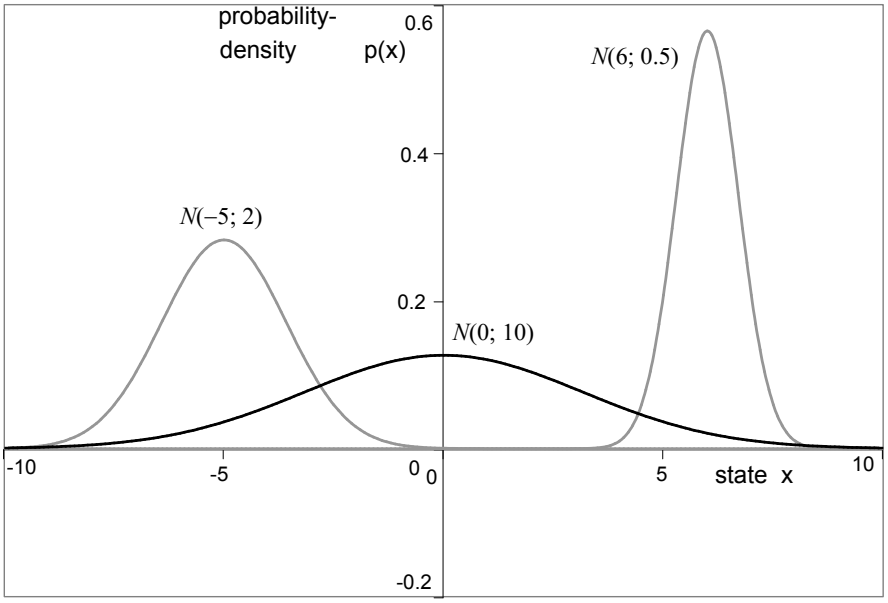


Figure 2.1: Gaussian distribution examples: Different means and standard deviation. The first parameters of the 2-tuples represent the mean, the second parameters the variance.

a single Gaussian. Advantages of normal distributions are their seclusiveness regarding addition and multiplication, i.e., the addition of two normally distributed random numbers leads to a pdf, that is also normal. Also, multiplication by a scalar (not a random variable) leads to another normal pdf. Since those operations can be executed very efficiently for normal distributions, algorithms based on normal distributions can be calculated very efficiently.

The properties of an n-dimensional Gaussian distribution $\mathcal{N}(\mu, \Sigma)$ are defined by the mean of the distribution (μ), and the covariance matrix, $\Sigma \in R^{n \times n}$. The first moment μ of a distribution corresponds to the expected value $E[X]$ of the distribution. The covariance matrix Σ is calculated by $E[(X - E[X])(X - E[X])^T]$. The density function $\mathcal{N}(\mu, \Sigma)$ of a n-dimensional normally distributed variable X can be calculated as follows:

$$p(X) = \frac{1}{\sqrt{(2\pi)^n \cdot \det(\Sigma)}} \exp^{-\frac{1}{2}(X-\mu)^T \Sigma^{-1}(X-\mu)} \tag{2.32}$$

The notation $\mathcal{N}(X; \mu, \Sigma)$ is often used to refer to the probability of X in equation (2.32). The expected value μ represents the most probable value of the distribution. In many applications this value corresponds to the estimated value of the object to model. The covariance matrix tells, how narrow or wide the whole distribution is, cf. Fig. 2.1, and if there are coherences between different dimensions of the state space. The covariances of a bivariate distribution can be visualized on a 2-d plane as an ellipse or a set of concentric ellipses, cf. Fig. 2.2. The covariances of a multivariate distribution can be

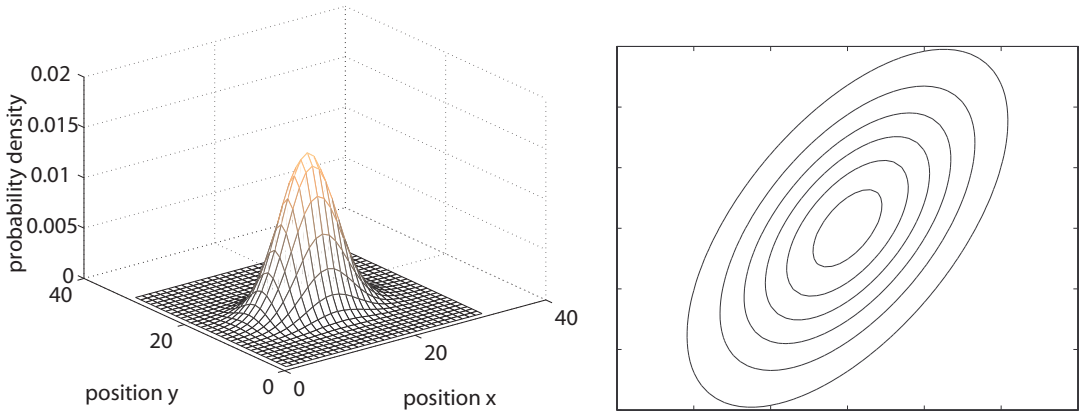


Figure 2.2: Two-dimensional Gaussian distribution: (a) 2-d shape of a Gaussian, (b) each trace depicts a set of equal values within the Gaussian function. The orientation of the ellipse denotes the correlation of the two variables.

visualized as a hyper ellipsoid [69].

2.3 Important Terms for Object Modeling

In the next sections the most common terms and concepts which are used for object modeling are discussed.

2.3.1 State Space

A state can be defined as a collection of all aspects of a robot's environment, that have an influence on the environment's future [114]. Certain variables can change over time, e.g., state variables considering the robot position. Other parameters, as such related to fixed objects stay unchanged. Furthermore, state spaces can be classified by the following criteria:

- Complete state: A state x_t is called *complete*, if it is the best predictor of the future [114], i.e., knowledge over its past and other sensory or control data would not lead to additional certainty for future predictions. Temporal processes fulfilling this requirement are called Markov chains. This criteria is hard to fulfill because usually only a part of all necessary variables, that influence an object state are known. Frequently, one has to take a subset of state variables. This subset is called an *incomplete state*.

- Discrete vs. continuous state space: Many classic problem solving techniques assume a *discrete* state space. One advantage of a discrete state space is that it can be interpreted as a graph, in which every node is related to a state and connected

to a finite number of other nodes. Just to mention, there exist infinite graphs as well which are not considered here. Having a finite number of states is realistic for a variety of problems, e.g., game trees. In games as Chess, Go, or Checkers only a limited number of game configurations is possible. However, for localization problems the state space is *continuous*. Sometimes discrete state spaces are used to approximate continuous ones for efficiency reasons, e.g., using grids, even though this approximation can decrease accuracy. Another example for discrete state variables are binary variables that represent the functionality of sensors or motors.

In [114] the following categorization of state variables is proposed:

- A *pose* is a tuple of variables, describing the position of a robot and its orientation related to a coordinate system. It can consist of six variables: three for the position within 3-d space plus three variables for the orientation. When robots move in 2-d space the number of dimensions can be reduced to three: two for the position plus one for the orientation.

- Another category consists of all variables, that can change the robot state. This includes the joint angles and every degree of freedom of the robot. Those variables are usually referred to as the *kinematic state* [114], or the robot *configuration* [102].

- The speed of a robot is called the *dynamic state* and can - as the six state variables for the position - consist of up to six variables.

- Another component consists of all those state variables referring to objects within the environment of the robot. Depending on the object, those values can be static, e.g., a hall without moving parts or dynamic, e.g., when modeling other mobile robots. Depending on the granularity of an object model, the number of necessary variables can go from just a few, to a high power of ten.

2.3.2 Further Modeling Terms

In addition to the definition of a state space further terms shall be introduced, which are of concern regarding object modeling.

- Markov chain: Complete states which are changing temporally are called *Markov chains*. Markov chains can be used whenever variables are conditionally independent. In object modeling Markov chains are used whenever the current state x_t is conditionally independent from all preceding states, given the state x_{t-1}:

$$p(x_t|x_{t-1}) = p(x_t|x_{t-1}, x_{t-2}, \ldots, x_{t-n}). \tag{2.33}$$

- Landmarks: Entities within the environment of a robot that are stationary are called *landmarks*. Landmarks may be either uniquely identifiable or ambiguous.

- Control actions change the state of the environment of a robot. The robot can actively assert forces onto its environment to change the position of objects or to change its own position. Also, the suppression of such forces can be called a control action. Thus, in [114] it is stated that a robot is performing control actions all the time.

- Odometry Data: A further kind of control data are odometry data. They have to measure the executed motion of a robot, e.g., the revolutions of the robot's wheels and give hint whereto the robot state has changed. Though odometers represent a form of sensory data, they are commonly interpreted as control action data.

- Belief: The internal estimation about the surroundings of a robot is called *belief*. Since the robot cannot measure the state of its position or the state of its environment directly, it has to make estimations using conditional probabilities. The probabilistic paradigm represents the robot's current belief by a probability density function over the space of all locations [114]. The belief over a state x_t will be denoted by $Bel(x_t)$. It is usually conditioned by the preceding sensory data and the preceding actions. In other contexts the belief is sometimes called *state estimate* or *estimate*.

- Environment sensor measurement: The robot uses its sensors to obtain information about the state of its environment. This can be, e.g., taking a camera image, a range scan or even using tactile sensors to receive information about the environment [114]. The result of such a perceptual interaction is usually called measurement, observation or percept. In this work we will use the term *percept* to describe a measurement vector of the robot position to a visually observed object, projected on the ground plane. To know, where to robot was looking at while perceiving an image, the joint angle information of the head camera joints are necessary. Thus, our percept is generated by visual and joint data. Furthermore, in this work the term *sensory data* usually refers to percepts, i.e., the distance vector from a robot to a perceived object.

- Sensor model: Throughout this work, we call the distribution function $p(z|x)$ a *sensor model*, where z denotes a measurement and x a state of the environment. It describes the probability for a given state x that a certain measurement z is taken.

2.4 Bayes Filters

Bayes filtering is widely used in object localization and tracking and one of its most important concepts [36]. It is used in cases of sensory noise and for sensor fusion whenever multiple sensors are available. Bayes theorem *(or Bayes law)* is applied for localization purposes, but not only for localization. Other approaches, i.e., color classification [106] or behavior planning use its results as well. Bayes filters are grounded on

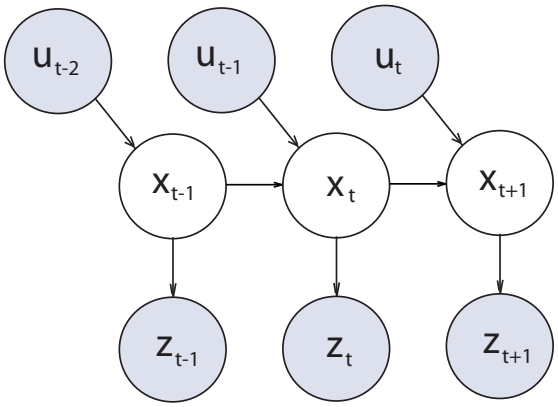


Figure 2.3: Bayesian network representation for a Hidden Markov model. The term *hidden* refers to the non-observability of the object state x_t. The robot performs actions u_t and takes sensory measurement z_t to estimate the object state x_t.

the Bayes law, which states that:

$$p(x|Z) = \frac{p(Z|x)p(x)}{p(Z)}, \tag{2.34}$$

where x is the object state and Z the set of measurements for this state. Bayes filters represent the current object state at time t by the probabilistic variable x_t. Variable z_t describes the measurement at time t. Since every robot can influence its environment, every state x depends on executed actions u as well. For every time t the object state x_t can be derived from the probability density function $Bel(x_t)$ over x_t [36]. Thus, $Bel(x_t)$ can be interpreted as the probability of x_t under the condition of preceding sensory information $z_1, \ldots, z_t$ as well as preceding actions $u_1, \ldots, u_{t-1}$.

$$Bel(x_t) = p(x_t|z_t, u_{t-1}, z_{t-1}, u_{t-2}, \ldots, z_1, u_0, z_0) \tag{2.35}$$

Bayes filters assume that currently executed actions u_t and measures z_t are conditionally independent from previous actions and measures, given the current object state x_t, as depicted in Fig. 2.3. Because the state cannot be observed directly, the Model underlying is usually referred to as *Hidden Markov model* (abbrv. *HMM*). A good introduction to HMMs and Bayesian networks can be found in [39]. With the Bayes law (2.34), equation (2.35) can be transformed into:

$$Bel(x_t) = \frac{p(z_t|x_t, u_{t-1}, \ldots, u_0, z_0)p(x_t|u_{t-1}, \ldots, u_0, z_0)}{p(z_t|u_{t-1}, \ldots, u_0, z_0)} \tag{2.36}$$

With the Markov assumption, Fig. 2.3, the conditional independency of z_t from the preceding measurement z, and actions u given current object state x_t we have:

$$Bel(x_t) = \frac{p(z_t|x_t)p(x_t|u_{t-1},\ldots,u_0,z_0)}{p(z_t|u_{t-1},\ldots,u_0,z_0)} \tag{2.37}$$

With the law of total probability this can be tranformed into:

$$= \frac{p(z_t|x_t)}{p(z_t|u_{t-1},\ldots,u_0,z_0)} \int p(x_t|x_{t-1},u_{t-1},\ldots,z_0)p(x_{t-1}|u_{t-2},\ldots,z_0)\mathrm{d}x_{t-1} \tag{2.38}$$

Using the Markov assumption, one gets:

$$Bel(x_t) = \frac{p(z_t|x_t)}{p(z_t|u_{t-1},\ldots,u_0,z_0)} \int p(x_t|x_{t-1},u_{t-1})Bel(x_{t-1})\mathrm{d}x_{t-1} \tag{2.39}$$

Now η is used as a normalizing factor, to make sure that the different probabilities within the state space sum up (or the pdf integrates) to 1:

$$Bel(x_t) = \eta p(z_t|x_t) \int p(x_t|x_{t-1},u_{t-1})Bel(x_{t-1})\mathrm{d}x_{t-1} \tag{2.40}$$

The resulting equation (2.40) is a recursive form of the Bayes filter. Thus, it is possible to calculate the new state estimation directly from the preceding state function. The equation can be split into two simpler equations. One equation *predicts* how the state estimation changes after the robot performed an action. The second equation *corrects* the predicted belief with incoming sensory data. If data from performed actions u_{t-1} is coming in, the new state is at first predicted. Therefore, this step is called *prediction step* or *time update step*:

$$Bel^{-}(x_t) \longleftarrow \int p(x_t|x_{t-1},u_{t-1})Bel(x_{t-1})\mathrm{d}x_{t-1} \tag{2.41}$$

Conversely, if new sensory data is available z_t the predicted state can be corrected. Therefore, this step is called *correction step* or *sensory update step*:

$$Bel(x_t) \longleftarrow \eta p(z_t|x_t)Bel^{-}(x_t) \tag{2.42}$$

To start the calculation at time $t = 0$, $Bel(x_0)$ has to be initialized with the available knowledge about x_0. If there is no prior knowledge available for x_0, a constant distribution over the state space can be assumed. Furthermore, the modeler has to define the sensor model $p(z_t|x_t)$, the system dynamics $p(x_t|x_{t-1},u_{t-1})$ and the representation form of $Bel(x_t)$. Thereby, the sensor model describes the probability of measurement z_t given a state x_t. The system dynamics, often referred to as process model describes changes of x_t after executing action u_{t-1} in state x_{t-1}. It shall be pointed out that u, x and z can be multidimensional vectors.

2.5 Summary

Bayes filters represent an abstract concept for world modeling. The interaction of a robot with its environment is modeled as a coupled system, in which the robot manipulates its environment by choosing controls and in which the robot can perceive its environment through its sensors [114].

The sensor model, the system dynamics and the form of belief representation are free to choose in the Bayes filter model. They have to be defined for a certain application, depending on the structure of the environment, calculational resources and required accuracy. The next chapter will discuss different kinds of recursive state estimation techniques, that are derived from the Bayes filter.

3 Bayesian Filtering Techniques

The various kinds of Bayesian filters differ presumably in the way they represent the belief function. One important class of Bayes filters are *Gaussian filters*, which represent the belief using one or multiple Gaussian functions. In [114] their contribution is stated as follows: "Gaussian filters constitute the earliest tractable implementations of the Bayes filter" and "They are also by far the most popular family of techniques to date - despite a number of shortcomings." One of the properties of Gaussians is their unimodality, i.e., they possess a single maximum and thus can be applied to many robot tracking problems, but are a poor match for many global estimation problems in which multiple distinct hypotheses exist.

In this chapter the Kalman filter, its advanced forms to nonlinear problems: Extended and Unscented Kalman filter (abbrv. *EKF, UKF*), and the Information filter are described. Later the functionality of other members within the Bayesian filter family are presented: the Particle filter, Grid-based approaches, and filters using topologies. Furthermore, optimization techniques regarding computational and memory needs are described.

3.1 The Kalman Filter and its Extensions

Kalman filters (abbrv. *KF*) [62, 38, 93, 119] are one of the most widely used form of Bayes filters. They use Gaussian distributions for the sensor and process model and can be implemented easily and efficiently. They are optimal for time discrete and linear dynamic processes, i.e., they minimize the squared error of the estimation. This property is lost when Kalman filters are used for non-linear processes. Much research has been done in the field of Kalman filters for non-linear processes [5], and on approximatively linearizing non-linear processes. With some extensions, which will be presented later in this chapter, Kalman filters can be applied to non-linear processes [32, 75, 1]. They also showed to be very efficient in high dimensional state spaces as in SLAM-applications [26, 76, 15, 22, 12]. Now the basic principles of a Kalman filter will be described.

Let $x_t \in \Re^d$ be the state to be modeled at time t, $\hat{x}_t$ is the estimation of x_t, and Σ_t the covariance matrix[1], estimating the a-posteriori modeling error. The Kalman model assumes that a state x_t can be calculated linearly from state x_{t-1} and the executed actions

[1] Σ is denoted as P in some literature

u_{t-1}:

$$x_t = Ax_{t-1} + Bu_{t-1} + \omega_{t-1} \tag{3.1}$$
$$\omega_{t-1} \sim \mathcal{N}(0, Q) \tag{3.2}$$

The state transition matrix A defines how the new state x_t at time t can be calculated, given the previous state x_{t-1}, not considering the control actions of the robot, that could change the current state. Those control actions u_{t-1} are taken into account by the control input matrix B. The process noise ω is assumed to be Gaussian white with a zero mean and with covariance matrix Q. The measurement z_t is assumed to be a linear function of the state x_t:

$$z_t = Hx_t + v_t \tag{3.3}$$
$$v_t \sim \mathcal{N}(0, R) \tag{3.4}$$

Matrix H transforms the state x_t into a measurement z_t which is affected by sensory noise v. The sensory noise v needs to be white as well, with zero mean, the covariance R and uncorrelated with ω.

In the first of two steps *prediction* and *correction*, as described earlier in Section 2.4, the Kalman filter predicts the new state $\hat{x}_t^-$ using the last estimated state $\hat{x}_{t-1}$ and the control actions u_{t-1}. Sensory data is not used for the prediction, that is why the estimation is called *a-priori estimation*[2].

So, $\hat{x}_t^-$ is the predicted value of x_t before receiving new sensory data z_t. Consequently x_t^- will be calculated from $\hat{x}_{t-1}$ and u_{t-1} as equation (3.1) describes:

$$\hat{x}_t^- \triangleq E[x_t | u_{t-1}, \ldots, z_0] \tag{3.5}$$
$$= E[Ax_{t-1} + Bu_{t-1} + w_{t-1} | u_{t-1}, \ldots, z_0] \tag{3.6}$$
$$= A\hat{x}_{t-1} + Bu_{t-1} \tag{3.7}$$

The transformation step to equation (3.7) takes into account that the expectation of ω is zero. The a-priori error covariance matrix, presented by matrix Σ^-, can be calculated accordingly:

$$\Sigma_t^- \triangleq E[(x_t - E[x_t])(x_t - E[x_t])^T | u_{t-1}, \ldots, z_0] \tag{3.8}$$
$$= AE[(x_t - E[x_t])(x_t - E[x_t])^T]A^T + E[\omega_t \omega_t^T] \tag{3.9}$$
$$= A\Sigma_{t-1}A^T + Q \tag{3.10}$$

The correction step is defined as the product from $p(z_t|x_t)$ and $Bel^-(x_t)$. The expected observation $\hat{z}_t$ at time t can be calculated as stated in equation (3.3):

$$\hat{z}_t = H\hat{x}_t^- \tag{3.11}$$

[2] a-priori stays for: "prior to the measurement", a-posteriori: "after the measurement"

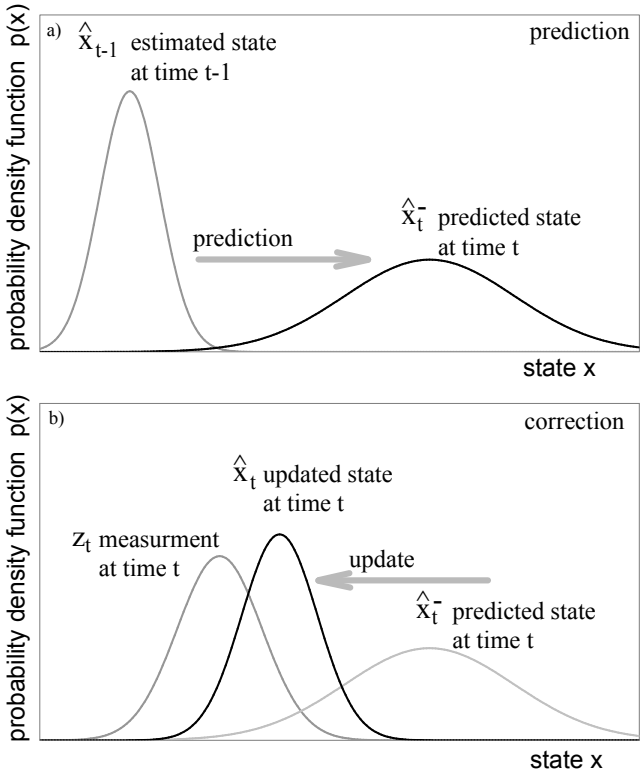


Figure 3.1: The Kalman filter loop: Two steps, *state prediction* and *correction* are illustrated using an one-dimensional Kalman filter. (a) After the prediction step the variance/covariances of the object state increase. (b) During the correction step they decrease.

The difference between the expected measurement $\hat{z}_t$ and the real measurement z_t is called *innovation*[3]. It tells, how strong the prediction of a state diverges from the measurement. Its covariance S_t is calculated from the sensor covariance R_t and the a-priori covariance of the state estimation Σ_t^-:

$$S_t = R + H\Sigma_t^- H^T \tag{3.12}$$

Now the Kalman gain can be calculated. The Kalman gain determines how strong the innovation, i.e., the difference between predicted state and measured sensory data will be considered for the a-posteriori state estimation $\hat{x}_t$. The bigger the Kalman gain is, the more the predicted state will be affected by the sensory data.

$$K_t = \Sigma_t^- H^T S_t^{-1} \tag{3.13}$$

With the Kalman gain the a-posteriori estimation $\hat{x}_t$ for state x_t and for the a-posteriori

[3]also "residual"

covariances Σ_t can be calculated:

$$\hat{x}_t = \hat{x}_t^- + K_t(z_t - \hat{z}_t) \tag{3.14}$$

$$\Sigma_t = \Sigma_t^- - K_t S_t K_t^T \tag{3.15}$$

The working steps of the Kalman filter are depicted in Fig. 3.1. If matrix R has big sensory data covariances, the sensory data have a lesser effect on the a-posteriori estimation than the prediction has. The covariances of the innovation will be bigger and result in a smaller Kalman gain. Thus, the deviation of the sensory data from the predicted object state will not be considered as strong as it would be for smaller values in R. It is clear that this enables a Kalman filter to be used especially for noisy sensory data. When R contains small values, the a-priori state estimation $\hat{x}_t^-$ will be corrected more by the sensory data z_t. In other words, Kalman filters with small sensor covariances in R, compared to the predicted measurement covariance $H\Sigma_t^- H^T$, react quicker on sensory data changes. For the filtering process the process covariance matrix Q and the sensory error covariance matrix R must be known. These matrices do not necessarily have to stay constant for every time step t. Moreover, the sensor matrix R can be generated by statistically analyzing the incoming sensory data on the robot for different scenarios. The process matrix Q represents the error, that occurs while propagating a state $\hat{x}_{t-1}$ into the next state $\hat{x}_t^-$. It is usually calculated using external measurements, e.g., ceiling cameras. The Kalman filter can also be used to model sensory data from different sources. In [110] is described, how to fuse multiple Gaussian distributions. A comparison of different possibilities for fusing a ball position is presented in [31]. Dietl and Gutmann described a possibility how to fuse sensory data of different robots in [25]. Therefore, they used a Kalman filter which incorporated sensory data from different robots. When using three robots this method proved to be robust to wrong sensory data, e.g., when one robot mistakenly perceives a ball in the image and the other two robots perceive correct data.

Another method to combine Gaussian beliefs, e.g., beliefs of different robots about an object is to create a Gaussian mixture model and to collapse the different components to a single Gaussian as demonstrated in [77]. Fig. 3.2 illustrates, that the approximation accuracy of the collapsed distribution depends on the distance of the mixture components to each other. The following theorem tells how to collapse a mixture of Gaussians [72]:

Theorem 3.1.1 *Let Q be the density function of a normalized mixture of n weighted Gaussians $\langle \omega, \mathcal{N}(X; \mu_i, \Sigma_i) \rangle$, where ω denotes the weights. Let P be a normal distribution $\mathcal{N}(X; \mu, \Sigma)$ defined as:*

$$\mu = \sum_i \omega_i \mu_i \tag{3.16}$$

$$\Sigma = \sum_i \omega_i \Sigma_i + \sum_i \omega_i (\mu_i - \mu)(\mu_i - \mu)^T \tag{3.17}$$

Then P has the same first two moments (means and covariances) as Q.

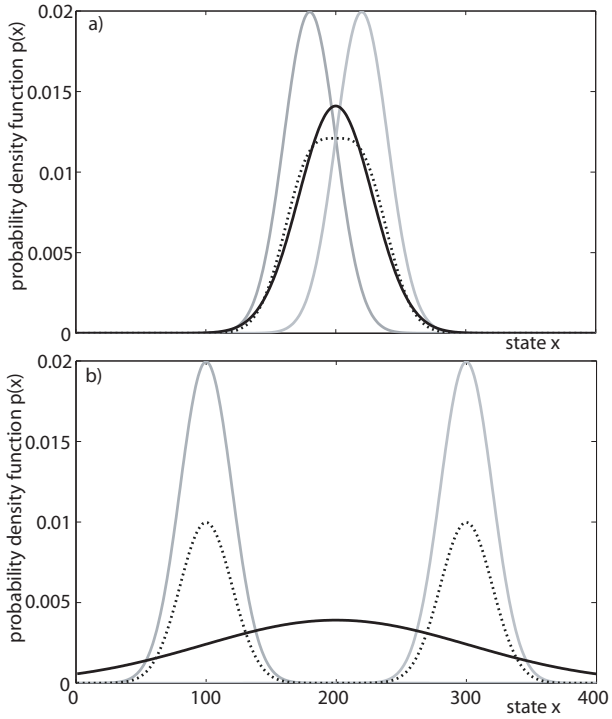


Figure 3.2: Collapsing Gaussian distributions: (a) Collapsing Gaussians (gray), that are close together produces a good approximation (black) to the normalized mixture distribution (dotted) of both source functions.
(b) When Gaussians (gray) are far apart, the collapsed Gaussian (black) is a poor approximator to the mixture (dotted) of the source functions.

The proof can be found in [77].

3.1.1 Extended Kalman Filter

The Extended Kalman filter (EKF) extends the normal Kalman filter (KF) to nonlinear problems. It has been widely used for self-localization and SLAM-problems [6, 14], whenever pure KFs cannot be applied. The assumption that observations are linear functions of the state and that the current state is a linear function of the previous state, is crucial for the correctness of KFs. Vice versa, if linearity is not given, KFs do not work optimal. In natural processes linearity is rare, so KFs are applicable for simple localization or tracking problems only.

The EKF relaxes the linearity assumption. State transitions are governed by nonlinear

functions, g and h:

$$x_t = g(u_t, x_{t-1}) + \epsilon_t \tag{3.18}$$
$$z_t = h(x_t) + \delta_t \tag{3.19}$$

Thus, EKFs are a generalization of KFs. The function g replaces the matrices A_t and B_t from equation (3.1), function h replaces matrix H from (3.3). Using arbitrary functions for g and h results in Non-Gaussians for the belief. With nonlinear functions g and h an exact state estimation is impossible. Moreover, the Bayes filter does not possess a closed-form solution.
The Extended Kalman filter approximates the resulting belief by a Gaussian. This is the similarity to Kalman filters, though for EKF this Gaussian is just an approximation of the real belief function in terms of estimating the mean and the covariances.

Now the focus lies on the linearization for the EKF. Linearization approximates the non-linear function by calculating the tangent g at the mean of the Gaussian. Even though this estimation can be erroneous, approximating function g using the tangent of one point showed to be highly computationally efficient. Once g is linearized, the original Kalman equations can be used. Also h is approximated using a tangent.

Taylor Expansion

Linearization in EKFs is done by first order Taylor expansion. The slope is given by the partial derivative:

$$g'(u_t, x_{t-1}) = \frac{\delta g(u_t, x_{t-1})}{\delta x_{t-1}} \tag{3.20}$$

The value and slope of g depend on its arguments. To linearize g' one has to chose the most likely state at the time of linearization, which is for Gaussians the mean of the posterior μ_{t-1}. So, g is approximated by the most likely state at u_t, the linear extrapolation is achieved by a term proportional to the gradient of g at μ_{t-1} and u_t [114] as equation (3.21) states:

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + \underbrace{g'(u_t, \mu_{t-1})(x_{t-1} - \mu_{t-1})}_{=:G_t} \tag{3.21}$$
$$= g(u_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1}) \tag{3.22}$$

The a-priori state probability can now be approximated as follows:

$$p(x_t|u_t, x_{t-1}) \approx \frac{1}{\sqrt{\det(2\pi Q_t)}} \exp^{-\frac{1}{2}m^T Q_t^{-1} m} \tag{3.23}$$

with

$$m := x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})$$

For a n-dimensional state space, matrix G_t is of size $n \times n$ and usually referred to as the *Jacobian*. The value of the Jacobian can change over time and for different types of u_t and μ_{t-1}.
For linearization of the measurement function h the Taylor expansion is developed around the most probable point $\overline{\mu_t}$ at the time it linearizes h:

$$h(x_t) \approx h(\overline{\mu_t}) + \underbrace{h'(\overline{\mu}_t)}_{:=H_t}(x_t - \overline{\mu_t}) \tag{3.24}$$

$$= h(\overline{\mu_t}) + H_t(x_t - \overline{\mu_t}), \tag{3.25}$$

with $h'(x_t) = \frac{\delta h(x_t)}{\delta x_t}$ one can estimate the Gaussian:

$$p(z_t|x_t) = \sqrt{\det(2\pi R_t)}\exp^{-\frac{1}{2}[z_t - h(\overline{\mu}_t) - H_t(x_t - \overline{\mu}_t)]^T R_t^{-1}[z_t - h(\overline{\mu}_t) - H_t(x_t - \overline{\mu}_t)]} \tag{3.26}$$

Now the full Extended Kalman filter algorithm [114] with μ_t and Σ_t can be written as:

$$\overline{\mu_t} = g(u_t, \mu_{t-1}) \tag{3.27}$$

$$\overline{\Sigma_t} = G_t\Sigma_{t-1}G_t^T + Q_t \tag{3.28}$$

$$K_t = \overline{\Sigma_t}H_t^T(H_t\overline{\Sigma_t}H_t^T + R_t) \tag{3.29}$$

$$\mu_t = \overline{\mu_t} + K_t(z_t - h_{\overline{\mu_t}}) \tag{3.30}$$

$$\Sigma_t = (I - K_tH_t)\overline{\Sigma_t} \tag{3.31}$$

3.1.2 Unscented Kalman Filter

Another method to approximate non-linear transformations is applied by the Unscented Kalman filter [60, 118]. It uses a weighted statistical linear regression. Thereby, for approximating g, the UKF deterministically extracts *sigma points* from the Gaussian and passes them through g.

3.1.3 Unscented Transform Linearization

The sigma points are usually located at the mean of the Gaussian and symmetrically along the main axes of the covariance, i.e., two sigma points per dimension. These $2n + 1$ sigma points $\chi^{[i]}$ of an n-dimensional distribution are typically selected as:

$$\chi^{[0]} = \mu \tag{3.32}$$

$$\chi^{[i]} = \mu + \left(\sqrt{(n+\lambda)\Sigma}\right)_i, \text{ for i = 1,..., n} \tag{3.33}$$

$$\chi^{[i]} = \mu - \left(\sqrt{(n+\lambda)\Sigma}\right)_{i-n}, \text{ for i = n + 1,..., 2n} \tag{3.34}$$

$$\tag{3.35}$$

The small index i or $(i-n)$ respectively, refer to the ith or $(n-i)$th row of the matrix square root[4]. The value of λ is determined by scaling parameters α, κ, and parameter n. They determine how far the sigma points are spread from the mean:

$$\lambda = a^2(n+\kappa) - n \tag{3.36}$$

Each sigma point $\chi^{[i]}$ has two associated weights, one weight $\omega_m^{[i]}$ for the mean and one weight $\omega_c^{[i]}$ for the covariance of the Gaussian, as follows [114]:

$$\omega_m^{[0]} = \frac{\lambda}{n+\lambda} \tag{3.37}$$

$$\omega_c^{[0]} = \frac{\lambda}{n+\lambda} + (1-\alpha^2+\beta) \tag{3.38}$$

$$\omega_m^{[i]} = \frac{1}{2(n+\lambda)}, \text{ for i = 1,..., 2n} \tag{3.39}$$

where β is a coefficient for encoding further knowledge of the distribution; $\beta = 2$ if the distribution is Gaussian. The sigma points are passed through function g, as a result one receives the mapped sigma points $\Gamma^{[i]}$:

$$\Gamma^{[i]} = g(\chi^{[i]}) \tag{3.40}$$

Now the estimated distribution function with μ' and Σ' can be calculated:

$$\mu' = \sum_{i=0}^{2n} \omega_m^{[i]} \Gamma^{[i]} \tag{3.41}$$

$$\Sigma' = \sum_{i=0}^{2n} \omega_c^{[i]} (\Gamma^{[i]} - \mu')(\Gamma^{[i]} - \mu')^T \tag{3.42}$$

The UKF approximation shows to be more accurate than the approximation of the EKF. Where the EKF is only accurate in the first term of the Taylor series expansion, the UKF is accurate in the first two terms [114], even though the EKF can be extended to be accurate for both two first Taylor expansions.

UKF Filtering Algorithm

Now the linearized distribution within the UKF shall be applied. Input and output are, as for the Kalman filter and EKF, the mean and the covariance matrix, further input

[4]Matrix A is a square root of matrix B, if $AA = B$.

data are the control actions u_t and the sensory data z_t:

$$\chi_{t-1} = (\mu_{t-1}, \mu_{t-1} + \gamma\sqrt{\Sigma_{t-1}}, \mu_{t-1} - \gamma\sqrt{\Sigma_{t-1}}) \tag{3.43}$$
$$\overline{\chi}_t^* = g(u_t, \chi_{t-1}) \tag{3.44}$$
$$\overline{\mu}_t = \sum_{i=0}^{2n} \omega_m^{[i]} \overline{\chi}_t^*[i] \tag{3.45}$$
$$\overline{\Sigma}_t = \sum_{i=0}^{2n} \omega_c^{[i]} (\overline{\chi}_t^* - \overline{\mu_t})(\overline{\chi}_t^* - \overline{\mu}_t)^T + Q_t \tag{3.46}$$
$$\overline{\chi}_t = (\overline{\mu}_t, \overline{\mu}_t + \gamma\sqrt{\overline{\Sigma}_t}, \overline{\mu}_t - \gamma\sqrt{\overline{\Sigma}_t}) \tag{3.47}$$
$$\overline{Z_t} = h(\overline{\chi}_t) \tag{3.48}$$
$$\hat{z}_t = \sum_{i=0}^{2n} \omega_m^{[i]} \overline{Z}_t^{[i]} \tag{3.49}$$
$$S_t = \sum_{i=0}^{2n} \omega_c^{[i]} (\overline{Z}_t^{[i]} - \hat{z}_t)(\overline{Z}_t^{[i]} - \hat{z}_t)^T + R_t \tag{3.50}$$
$$\overline{\Sigma}_t^{x,z} = \sum_{i=0}^{2n} \omega_c^{[i]} (\overline{\chi}_t^{[i]} - \overline{\mu}_t)(\overline{Z}_t^{[i]} - \hat{z}_t)^T \tag{3.51}$$
$$K_t = \overline{\Sigma}_t^{x,z} S_t^{-1} \tag{3.52}$$
$$\mu_t = \overline{\mu}_t + K_t(z_t - \hat{z}_t) \tag{3.53}$$
$$\Sigma_t = \overline{\Sigma}_t - K_t S_t K_t^T \tag{3.54}$$

In the beginning of the algorithm, the sigma points of the current distribution $\gamma = \sqrt{x + \lambda}$ are being determined. Then, in equation (3.44) these points are used to predict the state by using function g. Now the predicted mean and covariance matrix are computed from the resulting sigma points, as equations (3.45) and (3.46) show. To model the prediction noise uncertainty, Q_t is added to the sigma point covariance, where the prediction noise is assumed to be additive (3.46). A new set of sigma points is extracted from the predicted Gaussian in equation (3.49), which captures the overall uncertainty after the prediction step. Then a predicted observation is computed for all sigma points which are used to compute the prediction observation $\hat{z}$ and its uncertainty S_t. Matrix R_t is the covariance matrix for the additive measurement noise. S_t now represents the same uncertainty as $H_t\overline{\Sigma_t}H_t^T + R_t$ in equation (3.29) of the EKF algorithm [114]. The cross-covariance between state an observation is calculated in equation (3.51) and then used for calculation of the Kalman gain K_t. Now the final mean and covariance estimations can be calculated.

The calculation times of both, EKF and UKF have polynomial complexity in state dimension (typically cubic or possibly quadratic). For purely linear systems the estimates generated by UKFs are identical to those generated by Kalman filters [114]. EKFs and UKFs are powerful expansions to Kalman filtering techniques and still new developments are made. One of them are GP-Bayes filters, generating the process models in

every step from a training set by Gaussian process estimators [66], instead of using a fixed model.

A further example for the Kalman filter family is the *Information filter*.

3.1.4 Information Filter

The Information filter (abbrv. *IF*) works similarly to a Kalman filter. The main difference is the representation form of the Gaussian distribution. Kalman filters represent Gaussians by their mean μ and covariance matrix Σ. Information filters use a canonical parametrization. This representation form has some advantages and disadvantages which are discussed below.

Canonical Representation

Instead of a covariance matrix Σ^{-1}, the canonical representation uses the *information matrix* Ω, which is in fact the inverse of the covariance matrix:

$$\Omega = \Sigma^{-1} \tag{3.55}$$

The information matrix Ω is sometimes called *precision matrix* as well. The second parameter is the *information vector* ξ:

$$\xi = \Sigma^{-1}\mu \tag{3.56}$$

With Ω and ξ, the Gaussian is sufficiently defined.

Information Filter Algorithm

Now the Information filter algorithm is described and compared in each step to the Kalman filter algorithm. The Information filter model is somewhat equivalent to the Kalman filter model. Given the following matrices: state transition matrix A_t, the control input matrix B_t, measurement matrix C_t, sensory error matrix R_t and prediction error matrix Q_t, the Information filter model can be described as follows:

$$\overline{\Omega_t} = (A_t\Omega_{t-1}^{-1}A_t^T + Q_t)^{-1} \tag{3.57}$$

$$\overline{\xi}_t = \overline{\Omega}_t(A_t\Omega_{t-1}^{-1}\xi_{t-1} + B_t u_t) \tag{3.58}$$

$$\Omega_t = C_t^T R_t^{-1} C_t + \overline{\Omega}_t \tag{3.59}$$

$$\xi_t = C_t^T R_t^{-1} z_t + \overline{\xi_t} \tag{3.60}$$

Similarly to the Kalman filter, the Information filter is updated within two steps: the state prediction (3.57), (3.58) and the correction step (3.59), (3.60).

Duality of Kalman and Information Filter

At this point we want to take a closer look at the runtime complexity. The estimation step of the Information filter involves the inversion of two $n \times n$ matrices, which requires approximately $O(n^{2.4})$ [114]. However, in Kalman filters this step is additive and requires at most $O(n^2)$. During the correction step the computational complexities are reversed. Since measurement updates (corrections) are additive for the Information filter, they require at most $O(n^2)$, the Kalman filter has to invert matrices, which again has a time complexity of $O(n^{2.4})$. If each measurement is independent, the KF (or EKF, UKF or IF) equations can be computed as a sequence of independent scalar measurement updates - so typically, matrix inversion is avoided. When applied to robotics problems, the Information filter provides several advantages to Kalman filters: global uncertainty can easily be represented by simply setting $\Omega = 0$. When using moments one has to use infinite values for this kind of representation. This can become critical, especially when sensory measurements carry information about a strict subset of all state variables, which happens alot in robotics. Furthermore, the Information filter tends to be numerical more stable in many applications than the Kalman filter [114].

3.2 Multi-Hypotheses Tracking

In *Multi-Hypotheses Tracking*, (abbrv. *MHT*) the state is represented by a mixture of Gaussians [2, 3, 101, 58, 3], where each Gaussian component with index i has a mean $\mu_{t,i}$, covariance $\Sigma_{t,i}$, and the mixture weight $\omega_{t,i}$, which determines the contribution of the component to the posterior:

$$Bel(x_t) = \frac{1}{\sum_i \omega_{t,i}} \sum_i \omega_{t,i} \frac{1}{\sqrt{\det(2\pi\Sigma_{t,i})}} \exp^{-\frac{1}{2}(x_t-\mu_{t,i})^T \Sigma_{t,i}^{-1}(x_t-\mu_{t,i})} \tag{3.61}$$

An example is given in Fig. 3.3. Due to their ability to represent multi-modal distributions, MHT approaches are able to solve the global localization problem problem, or the "kidnapped robot problem" [101, 58], where a robot has to localize within its environment without prior knowledge. Since each hypothesis is represented by a Gaussian, it relies on the assumptions of usual Kalman filters. However, in case of nonlinear models, multiple EKFs or UKFs can be used to represent and to model the hypotheses. Suppose the prior distribution is a mixture of m^- Gaussians, and the measurement distribution requires a mixture of m_z Gaussians. Then the posterior distribution is a mixture of $m^+ = m^- \cdot m_z$ Gaussians. Therefore, clearly if we try this approach exactly, we get an exponential explosion in the number of mixture components. This is where main heuristics are needed, to determine in which cases hypotheses have to be pruned or merged, as described in [3] to reduce the number of mixtures. It is clearly substantially more computationally demanding than single hypothesis KF based algorithms. The underlying assumptions of near Gaussian, white noise are usually still required.

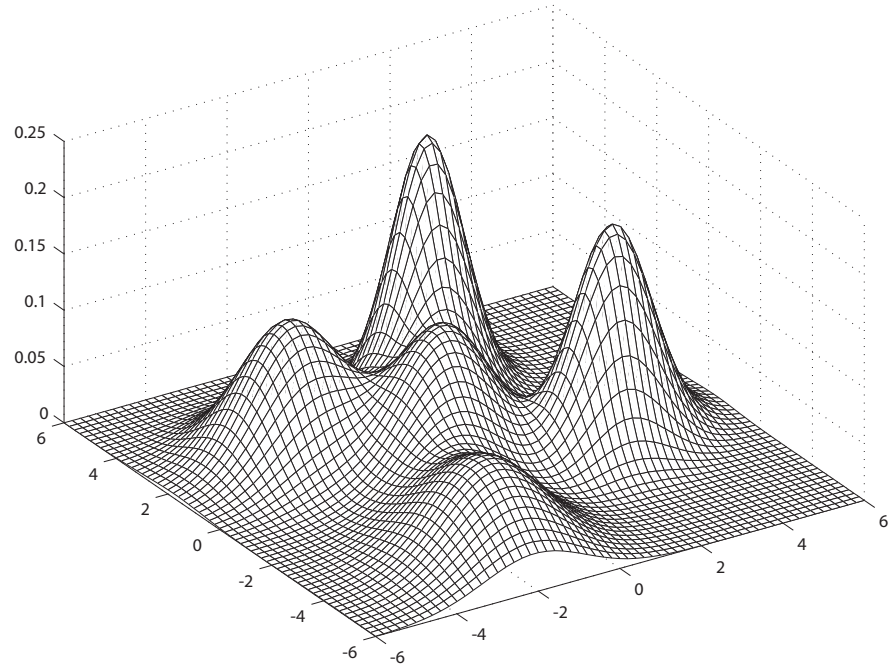


Figure 3.3: Gaussian mixtures: A two-dimensional distribution consisting of five Gaussians is presented.

3.3 Grid-Based Localization

Grid-based localization approaches represent the belief function by a piecewise constant function. Some sources also refer to it as the Histogram filter [114]. For indoor localization the spatial resolution of the grids is usually between 10 and 40 cm, the angular grid resolution is usually 5 degrees [32]. Grid-based approaches were successfully used in an office building, described in [10] and [9]. One advantage of grid-based approaches is their ability to represent arbitrary distributions, and thus to solve the global localization problem. Imagine a robot facing a wall on a long corridor, the robot could be localized on a very long and thin line within this corridor, which can easily be represented by a grid. In contrast to topological approaches, the metric approximations provide accurate position estimates in combination with high robustness to sensory noise [32]. Grid based approaches were also applied to the museum tour guide-robots Rhino and Minerva [23]. However, their ability to represent arbitrary distributions comes at high computational and memory costs. In case of three-dimensional localization, the robot has to keep the three-dimensional grid in memory and it has to update it in every step and for each observation. A variety of methods is available to reduce the computational and memory complexity, using more efficient sensor models, selective update schemes, and adaptive tree based representations, thus enabling the robot to run grid-based localization techniques under real-time conditions. Clearly,

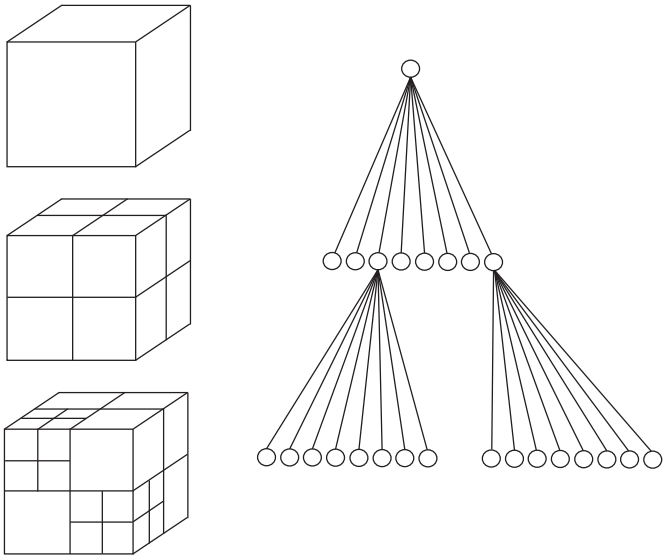


Figure 3.4: 3-d grid represented by an hierarchic octree: Each level within the octree encodes a certain grid resolution. Leave nodes can contain position probabilities or - in case of occupancy grids - occupancy encodings.

the exponentially growth in complexity with the number of dimensions is hindering grid-technologies to be used in higher-dimensional state spaces.

In the SLAM domain, grid-based techniques were applied in combination with occupancy grids [13]. Thereby, a grid is used to represent a map of the robot's surroundings. Each grid cell contains only boolean information for occupied or non-occupied space, usually "1" for "occupied" and "0" for "free". The most common domain of occupancy grid maps are 2-d floor plans, which describe a 2-d slice within the 3-d world, whenever the robot navigates on a flat surface [114]. They can be generalized to 3-d representations but at significant computational expenses.

Distribution Function Encoding with N-Trees

A n-tree can efficiently store a hierarchic grid, i.e., a grid with different resolutions at different areas with the state space. This can be useful during the localization process whenever some parts of the map are unknown and other parts are well known. Also for encoding a particle distribution, n-trees can be applied, which was demonstrated with quad-trees in [87]. The branching level of an n-tree is usually 2^n, where n is the dimension of the state space. An example is shown in Fig. 3.4.

3.4 Topological Approaches

Topological approaches use symbolic, graph structures. Each node within the graph represents a certain position within the robot's surroundings. Thus the state space is discrete and consists of a set of locally distinctive locations such as corners or crossings of hallways [13, 50, 18, 68]. Topological approaches are very efficient and can represent arbitrary distributions over the discrete state space. Thus, they can solve the global localization problem. They can also be used for high-dimensional state spaces, because the state dimensionality does not necessarily affect the complexity of the topological structure [32]. A key disadvantage is the coarseness of the representation. Position estimations do provide inaccurate data about the robot position only. Another disadvantage is the fact that the sensory data must be related to the symbolical structure of the graph. So it could happen that adequate features are not available in arbitrary environments [32].

Algorithm 1: The basic particle filter algorithm

Input: $S_{t-1} = \{\langle x_{t-1}^{(i)}, \omega_{t-1}^{(i)} \rangle | 1, \ldots, n\}$ representing belief $Bel(x_{t-1})$, control measurement u_{t-1}, observation z_t

```
1  S_t := ∅, α := 0                          // Initialization
2  for i := 1 to n do
3  |                                         // Resampling: Draw state from previous belief
4  |  Sample an index j from the discrete distribution given by the weights in S_{t-1}
5  |                                         // Sample: Predict next state
6  |  Sample x_t^(i) from p(x_t|x_{t-1}, u_{t-1}) conditioned by x_{t-1}^(j) and u_{t-1}
7  |  ω_t^(i) := p(z_t|x_t^(i))              // Compute importance weight
8  |  α := α + ω_t^(i)                       // Update normalization factor
9  |  S_t := S_t ∪ {⟨x_t^(i), ω_t^(i)⟩}      // Insert sample into sample set
10 end
11 for i := 1 to n do
12 |  ω_t^(i) := ω_t^(i) / α                 // Normalize weights
13 end
14 return S_t
```

3.5 Monte-Carlo Localization

A widespread member of the Bayes filter family, is the *Monte- Carlo Localization* (abbrv. *MCL*), also referred to as the particle filter algorithm. It has become one of the most important localization algorithm in robotics - and many different kinds of particle filter approaches have been developed [27]. MCL uses a *particle set* for belief representation

Bel_{x_t}, consisting of M particles $\chi = x_t^{[1]}, \ldots, x_t^{[M]}$. A particle[5] is usually an element of the state space, but can also be an element of a subspace of the state space.

The basic Monte-Carlo Localization algorithm as described by Fox [32] is presented in Alg. 1. In the beginning the particles are resampled from the previous belief, according to their weights. Then the particles are updated by the motion model (prediction step). Now the measurement model is applied to determine the importance weights of each particle. The initial belief Bel_{x_0} is usually generated by randomly distributing M particles with prior belief over the state space and assigning the uniform importance factor n^{-1} to each particle.

3.5.1 Properties of MCL

One of the biggest advantages of MCL approaches is their ability to model almost any distribution of practical importance. In contrast to other filters as KF it is not bound to certain representation functions as Gaussians which are limiting the representational power. With the number of particles M one can trade modeling accuracy for computational efficiency and vice versa. The advantage of MCL being non-parametric also leads to its ability to model multi-modal probability distributions. In contrast to MHT techniques one does not have to generate or destroy hypotheses for the different objects to track. However, in its present form, MCL cannot recover from robot kidnapping or global localization failures. This is because particles away from the most likely position disappear over time and only those nearby the probable position remain, which is known as *particle depletion*. As a result there are no more particles at a position where the robot is kidnapped to, making it impossible to recover from a wrong position.

The problem is especially relevant, when the number of particles is relatively small or if the state space is large. It can be solved by adding random samples to the particle distribution in each time step. Of course the number of particles stays constant, but some elements of the particle set are randomly repositioned. The mathematical justification for this *injection of random particles* is the small possibility for the robot being kidnapped. The question remains, of how many particles have to be added to the distribution. Adding a fixed number in every step would be possible but a better solution is to adjust the number of particles to add by an estimation of the local performance. One could monitor the probability of sensory measurements [114]:

$$p(z_t|z_{1:t-1}, u_{1:t}, m) \tag{3.62}$$

and compare it to the average measurement probability, learned from data. In particle filters one can easily calculate the average measurement probability from the importance weights:

$$\frac{1}{M}\sum_{m=1}^{M} \omega_t^{[m]} \approx p(z_t|z_{1:t-1}, u_{1:t}, m) \tag{3.63}$$

It is useful to average the probability of sensory measurements over several time steps.

[5] also called "sample"

There exist multiple reasons for a low measurement probability, besides a localization failure: unnatural high noisy sensory data, spread out particles during global localization.

There are also extensions to MCL which generate new particles not just arbitrarily on the field but in accordance to the measurement distribution. These measures can be very useful when only a limited number of particles is available. Just to mention, an adaptive variant of MCL, that determines the number of random particles by short-term and long-term likelihood of sensory measurements is the *Augmented MCL* [114, 80].

3.5.2 Adaptive Particle Filtering

As stated in [114] the number of particles is a crucial parameter in particle filters. The time complexity of the basic update of the particle filter algorithm (prediction and correction without clustering or KLD-sampling) is linear in the number of samples needed for the estimation [33]. To avoid particle depletion (too few particles in certain areas while the particle set converges to another area), particle filters require to maintain a high number of particles. In [24] 20000 particles, and in experiments in [33] 100000 particles were used. But in the convergence region, more particles than necessary are available, causing unnecessary calculation. Depletion can be avoided using "auxiliary particles". An approach to reducing the number of particles in regions with a high particle density and keeping it in other regions is called "KLD-sampling", as it is using the Kullback-Leibler divergence measure (abbrv. *KLD*) for estimating the number of particles necessary for a belief representation. Before describing KLD-sampling, another existing approach called Likelihood-based adaptation shall be introduced.

Likelihood-Based Adaptation

This approach determines the number of particles based on the likelihood of observations. The approach generates samples until the sum of the non-normalized likelihoods exceeds a pre-specified threshold [33]. The assumption behind this approach is as follows: If the particle set matches the sensor reading, each particle weight is high and few particles are necessary. If on the other hand, the particles are not in tune with the sensor readings, the particle weights are lower and so is the uncertainty about the robot's position. As a result, more particles are created. This idea relates to the property that the variance of the importance sampler is a function of the mismatch between target distribution and the proposal distribution. But this mismatch is not always a good indicator for the necessary number of particles [33]. Given an example in which a robot can localize itself within a square shaped room. If the robot perceives a corner, its particle distribution converges to all four corners of that room. One has an ambiguous proposal distribution, where each particle has a high weight. However, the number of necessary particles to represent such a distribution should obviously be higher than if all particles converged to a single position. The likelihood based approach proved to be superior to particle filters with fixed sample sets as has successfully been demonstrated

in mobile robotics [32] and dynamic Bayesian networks [67], but for the given reasons this approach does not fully exploit the possibilities of adopting the size of sample sets.

KLD-Sampling

The assumption of KLD-sampling is that at each iteration of the particle filter the number of particles is determined in such a way that with probability $d - \delta$ the error of the true posterior and the sample based approximation is less than ε [32]. Assume that the true posterior is given by a discrete, piecewise constant distribution as a discrete density tree or multi-dimensional histogram, as in [90, 67, 83]. At first could be demonstrated that given this representation, the number of particles can be determined by the distance between the sample based maximum likelihood estimate (MLE) and the true estimate does not exceed a given threshold ε. The distance between the (MLE) and the true posterior is measured by the KL-distance [19]. The KL-distance is the measure between two probability distributions p and q and can be calculated as:

$$K(p,q) = \sum_x p(x) \log \frac{p(x)}{q(x)} \tag{3.64}$$

The KL-distance is never negative and it becomes zero if and only if both distributions are identical. It is not a metric, as it is not symmetric. The triangle property also does not hold but it is accepted as a measure for the difference between probability distributions. At first one has to determine the number of particles to have a good approximation of an arbitrary, discrete probability distribution [96]. Now it can be described how to modify this distribution, to realize this adaptation approach. Fox assumes in his work [33] that n samples are drawn from a distribution with k different bins. The vector $\underline{X} = (X_1, \ldots, X_k)$ shall denote the number of samples drawn from each bin. Thus $\underline{X}$ is distributed according to a multinomial distribution, i.e., $\underline{X} \sim multinomial_k(n, \underline{p})$, where $\underline{p} = p_1, \ldots, p_k$ describes the probabilities of each bin [32]. The maximum likelihood estimate of $\underline{p}$ using the n samples is given by $\underline{\hat{p}} = n^{-1}\underline{X}$. The likelihood ratio statistic λ_n for testing $\underline{p}$ is:

$$\log \lambda_n = \sum_{j=1}^{k} X_j \log \left(\frac{\hat{p}_j}{p_j} \right) \tag{3.65}$$

With $X_j = n\hat{p}_j$ this can be written as:

$$\log \lambda_n = n \sum_{j=1}^{k} \hat{p}_j \log \left(\frac{\hat{p}_j}{p_j} \right) \tag{3.66}$$

With (3.64) and (3.66) one can see that the likelihood ratio statistics is n times the KL-distance K between the MLE and the true distribution:

$$\log \lambda_n = nK(\underline{\hat{p}}, \underline{p}) \tag{3.67}$$

It can be proven that the likelihood ratio converges to a chi-square distribution with k-1 degrees of freedom [32, 96]:

$$2\log\lambda_n \rightarrow_d \chi^2_{k-1} \text{ as } n \rightarrow \infty \tag{3.68}$$

Fox then introduces $P_{\underline{p}}(K(\underline{\hat{p}}, \underline{p}) \leq \varepsilon)$ to denote the probability that the KL-distance between the true distribution and the sample-based MLE is less or equal to ε. Now the relationship between this probability and the number of necessary particles can be derived:

$$P_{\underline{p}}(K(\underline{\hat{p}}, \underline{p}) \leq \varepsilon) = P_{\underline{p}}(2nK(\underline{\hat{p}}, \underline{p}) \leq 2n\varepsilon) \tag{3.69}$$

$$= P_{\underline{p}}(2\log\lambda_n \leq 2n\varepsilon) \tag{3.70}$$

$$\doteq P(\chi^2_{k-1} \leq 2n\varepsilon) \tag{3.71}$$

Equation (3.71) is derived from (3.67) and the convergence result in (3.68). The quantiles of the chi-square distribution are given by:

$$P(\chi^2_{k-1} \leq \chi^2_{k-1,1-\delta}) = 1 - \delta \tag{3.72}$$

Now chose n in such a way that $2n\varepsilon$ is equal to $\chi^2_{k-1,1-\delta}$, by combining (3.71) and (3.72) one gets:

$$P_{\underline{p}}(K(\underline{\hat{p}}, \underline{p}) \leq \varepsilon) \doteq 1 - \delta \tag{3.73}$$

As a result, the number of particles necessary to achieve a certain approximation quality can be determined. Concluding, if one chooses a number of samples n as:

$$n = \frac{1}{2\varepsilon}\chi^2_{k-1,1-\delta} \tag{3.74}$$

then with probability $1 - \delta$ the KL-distance between the MLE and the true distribution is less than ε (see (3.73)). To solve equation (3.74) one has to compute the quantiles of the chi-square distribution. Therefore, Fox uses the Wilson-Hilferty transformation [59] as an approximation:

$$n = \frac{1}{2\varepsilon}\chi^2_{k-1,1-\delta} \doteq \frac{k-1}{2\varepsilon}\left(1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}}z_{1-\delta}\right)^3, \tag{3.75}$$

where $z_{1-\delta}$ is the upper $1 - \delta$ quantile of the standard normal distribution [32]. The values of $z_{1-\delta}$ for typical values δ are readily available in standard statistical tables. As one can see in (3.75), the required number of particles is proportional to the inverse of the error bound ϵ. Now will be presented, how the KL-distance can be used in particle

filters.

Algorithm 2: KLD-sampling applied in Monte-Carlo Localization

Input: $S_{t-1} = \{\langle x_{t-1}^{(i)}, \omega_{t-1}^{(i)} \rangle | 1, \ldots, n\}$ representing belief $Bel(x_{t-1})$, control measurement u_{t-1}, observation z_t, bounds ε and δ, bin size Δ, minimum number of samples $n_{\chi_{min}}$

1 $\overline{\chi}_t := \chi_t := \emptyset$;
2 $S_t := \emptyset, n := 0, n_\chi := 0, k := 0, \alpha := 0$ *//Initialization*
3 **while** $(n < n_\chi$ **or** $n < n_{\chi_{min}})$ **do**
4 Sample an index j from the discrete distribution given by the weights in S_{t-1}
5 Sample $x_t^{(n)}$ from $p(x_t | x_{t-1}, u_{t-1})$ using $x_{t-1}^{(j)}$ and u_{t-1}
// Predict next state
6 $\omega_t^{(n)} := p(z_t | x_t^{(n)})$ *// Importance weights*
7 $\alpha := \alpha + \omega_t^{(n)}$ *// Update normalization factor*
8 $S_t := S_t \cup \{\langle x_t^{(n)}, \omega_t^{(n)} \rangle\}$ *// Insert sample into sample set*
9 **if** *($x_t^{(n)}$ falls into empty bin b)* **then**
10 k:=k+1 *// Update number of supported bins*
11 b:=non-empty *// Mark bin*
12 **if** $(n \geq n_{\chi_{min}})$ **then**
13 $n_\chi := \frac{k-1}{2\varepsilon} \left(1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}} z_{1-\delta}\right)^3$ *// Update member*
// of desired samples while KL-bound is not reached
14 **end**
15 **end**
16 **end**
17 **for** $i := 1$ *to* n **do**
18 $\omega_t^{(i)} := \omega_t^{(i)} / \alpha$ *//Normalize weights*
19 **end**
20 return S_t

Application in Particle Filters

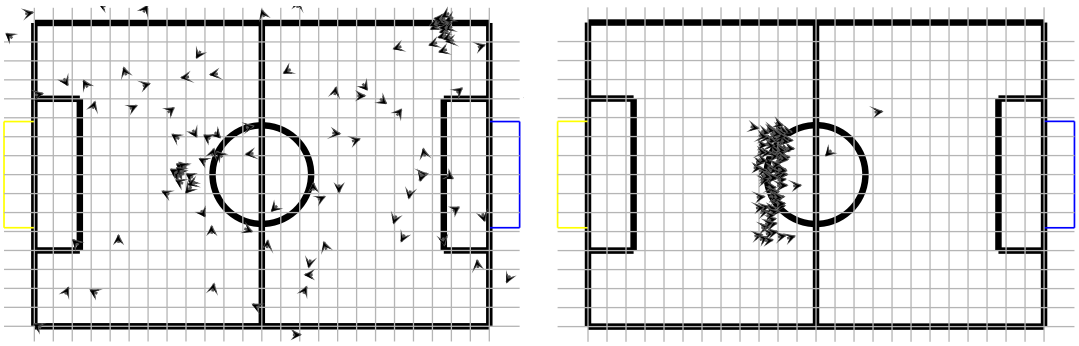
In contrast to the presented derivation of the KLD-measure, the true posterior is usually unknown when using a particle filter. As an alternative, Fox relies on the sample based representation of the predicted belief to estimate the posterior, which is generated in line 5 of Alg. 2. Furthermore, equation (3.75) indicates that it is not necessary to know the complete distribution, and that k is already sufficient to determine the number of supported bins. The value of k can be estimated by counting the number of support bins during sampling, so knowing the quantity of supported bins beforehand is not necessary. Alg. 2 describes that the number of supported bins is updated after the generation of each sample.

Furthermore, a check is conducted to determine if the minimum number of particles

$n_{\chi_{min}}$ was reached. Parallel to that, the number of generated samples n is increased as well. Thus, in early stages of sampling, k increases with almost every new generated sample, since all new samples fall into empty bins. This increase in k results in an increase of desired samples n_χ. Later, more and more bins are non-empty, so n_χ increases only occasionally. Because the number of generated samples n increases in every step, n will finally reach n_χ and sampling is stopped [33].

3.6 Distribution Function Measures

As presented in the upper section, Bayes filters can be implemented with a variety of different belief representations, e.g., Gaussians, particles, grid cells, which all have advantages and disadvantages. If it comes to communication of beliefs, encoding a Gaussian is easy, in a n-dimensional space, one has to communicate n elements for the mean vector and n^2 elements for the covariance matrix. Imagine a particle set with m particles, one could communicate every particle with n elements in each particle, so $n \cdot m$ elements had to be communicated. Thus, the communication of particle distributions can be computationally expensive, especially for big particle sets. A similar problem occurs when communicating n-dimensional grids, the number of grid cells can become very large. Another question is how to estimate the accuracy of the particle/grid distribution. For Gaussian distributions the answer can be found easily within the covariance matrix. For multiple Gaussians it is harder to judge the accuracy. Also for particle filters it is more complicated to find an appropriate convergence- or accuracy measure.



(a) (b)

Figure 3.5: Particle convergence examples, 100 particles: (a) Particle distribution of a robot having no prior knowledge about its position - particles are more or less equally distributed, (b) particle distribution converged after perceiving sensory data.

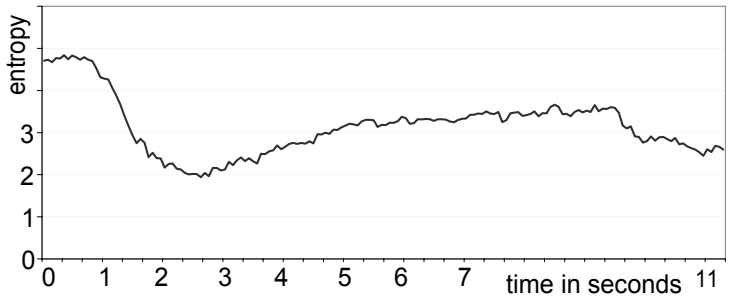


Figure 3.6: Entropy-time-diagram for a sample particle distribution from Fig. 3.5. Given 100 particles, 384 grid cells. At time t = 0, with no prior information the particles were equally distributed throughout the state space, after one second the particle set converged.

Entropy as a Convergence Measure

The entropy calculation of a particle distribution can be useful to describe its convergence and its ambiguity. It has several advantages compared to other measures as covariances, one advantage is its ability to measure the convergence of arbitrary distributions. The following example presents a robot, that self-localizes on a soccer field. The particle distribution is depicted at the beginning and at the end of the self-localization process, cf. Fig. 3.5. Its entropy function over time is presented in Fig. 3.6. The particle entropies were calculated using equation 2.30, we get:

$$H_p(x) = \sum -p(\frac{n_i}{n_c}) \log p \left(\frac{n_i}{n_c} \right) \tag{3.76}$$

for the number of all particles $n_c = 100$ particles and n_i particles in each grid cell i.

3.7 General Discussion

The state estimation problem usually contains the problem of estimating the position of an object or the position of the robot itself within a given map of the robot's environment. Bayesian filters have been successfully applied to a variety of localization problems. Depending on the modeling task, different types of Bayesian filters are appropriate. Kalman filters are comparably easy to implement but can only be used for unimodal distributions and linear dynamic processes. Extensions to handle non-linear dynamic processes are the Extended and Unscented Kalman filters. For multimodal distributions Multi-Hypotheses Tracking was developed. For arbitrary distributions grid-based and Monte-Carlo Localization approaches can be used. Especially Monte-Carlo Localization has become very popular, even though it can become computationally demanding, depending on the number of samples involved. Therefore, a variety of optimization techniques as KLD-sampling have been developed.

With those basics in mind, the next chapter takes a closer look on how multiple agents can cooperatively model an object state and how to cooperatively self-localize through communication of percept data.

4 Spatial Relationships of Visual Sensory Data

In this chapter possibilities of cooperative world modeling based on data exchange between different agents are described. The first section discusses, which kind of information is applicable for data exchange. Therefore, a closer look at different kinds of sensory data is taken. In the following sections will be described how this information can be integrated into a model.

4.1 Sensory Data

Robots perceive their surroundings with sensors. Usually the sensors are carried with the robot, even though there are scenarios, where sensors are not moving with the robots. When a robot is perceiving data from objects within its environment, it can immediately calculate the position vector of the object relative to the sensor. Those measurement vectors are called *percepts*. Percepts can have different properties, depending on the sensors and thus, different accuracies, cf. Fig. 4.1. Picture based sensors can measure the angle to features within the image. To measure a distance to a known object, one has to consider different of those angles. If the size of the object is unknown, other approaches, e.g., using the angle between the object and the horizon can be applied.

Laser scanners do not have this limitation. They can measure the angle and the distance to a point of their environment by calculating the running time of a laser beam from the emitter back to the sensor. Still, laser scanners and visual sensors can usually only perceive a small part of their surroundings. Some visual sensors use sophisticated mirrors to widen the field of view to have an *omnidirectional* sensor [99]. The robots that were used in this work have visual sensors, so the focus of this work lies on the incorporation of vision based sensory data. In this work the terms *visual sensory data* and *percepts* are used equivalently. Percepts are often considered to be independent of each other to simplify computation, even if they are used for the same purpose, e.g., for localization [97]. Using the distance of features detected within a single camera image to improve Monte Carlo Localization was proposed by [63]: When two landmarks are detected simultaneously, the distance between them yields information about the robot's whereabouts. Considering the horizontal order of percepts within an image was used for localization by Wagner et al. in [117]. The concept of ordering information was introduced earlier in [103]. It must be pointed out that sensory data need a frame of reference, a coordinate system in which the sensory data is represented. A possible coordinate frame for the percepts is to use the frame of the perceiving robot. Another possibility is to use a reference system within the robot's surroundings. The

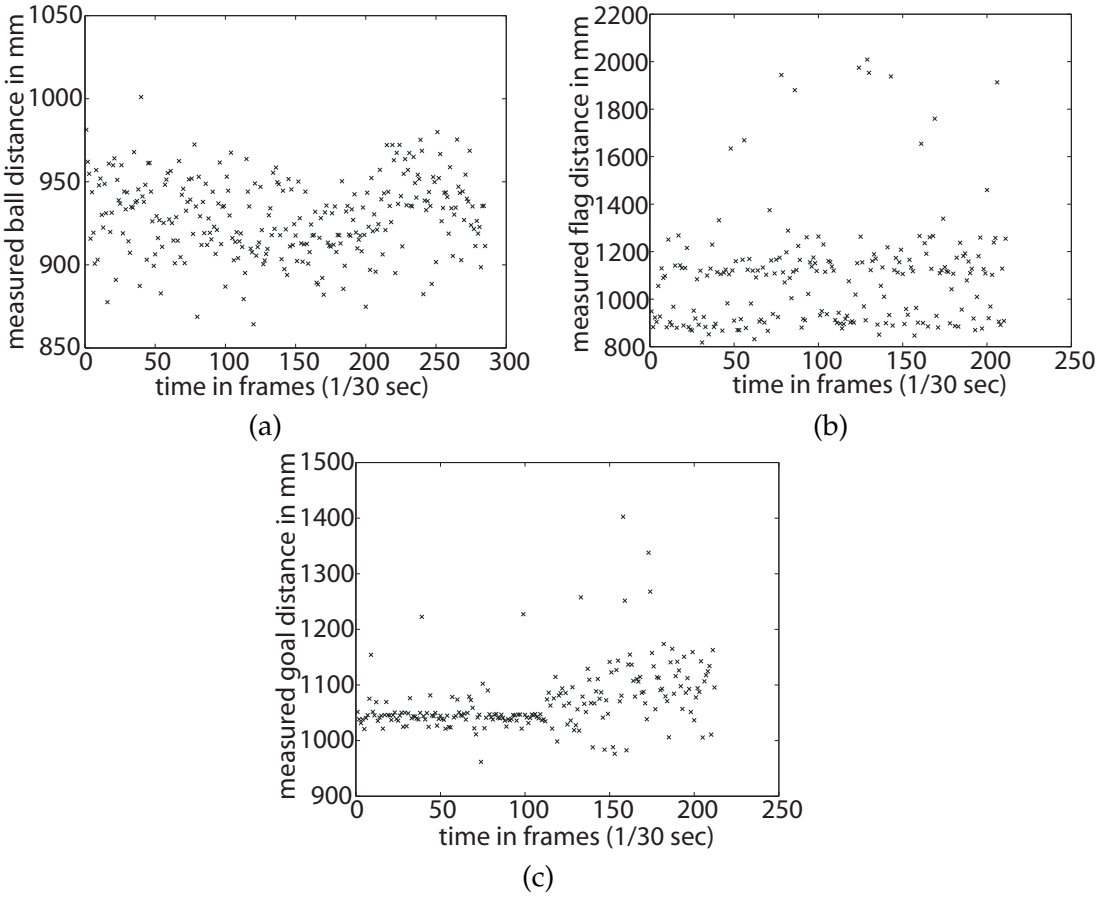


Figure 4.1: Sensory data variance, object distance: 1000 mm. (a) Measured distance to a ball, (b) measured distance to a flag and (c) to a goal.

transformation between different reference systems can lead to increased modeling errors, especially when sensory data is noisy [29]. Since most percepts use the perceiving robot as reference system, it can be useful to apply those reference systems to the object model as well. Those models, where the perceiving robot or the sensor is the source of the reference systems are usually called *egocentric*, or *local* models [34], see Fig. 4.2.

4.2 Reference Systems for World Modeling

Within mobile robotics another kind of reference systems exist, which is called *allocentric* or *global*. This method is frequently used in multi-agent systems so that all agents refer to the same coordinate frame. Within the next two sections the advantages and disadvantages of both methods for representing an object state are discussed.

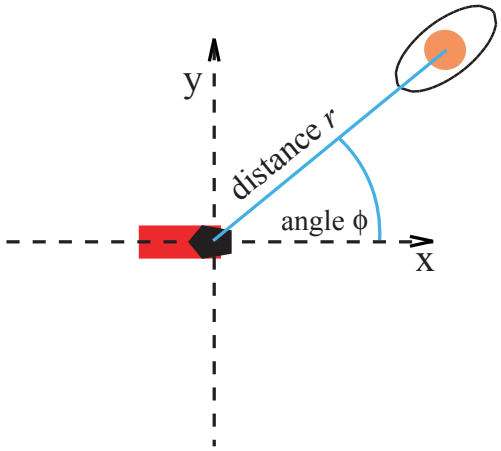


Figure 4.2: Percept and its distance/angle covariance form: The ellipse depicts the covariance matrix of the percept.

4.2.1 Egocentric World Modeling

One advantage of egocentric models is the abstraction of many variables that are necessary when using an allocentric frame of reference. The velocity of an object relatively to a robot for example, can be calculated without further knowledge about the position of the robot. However, using egocentric reference systems make it harder for a moving robot to model the speed of an object relative to the ground. But in many cases knowing the relative speed to the robot is already sufficient. Whenever the positions of objects have to be interchanged between different robots, e.g., for cooperative behavior like carrying an object together, egocentric models can become disadvantageous. Furthermore, the motion data of a robot, especially of a legged robot can be uncertain. Noisy motion data are usually caused by slippage, traction loss, or collisions [92, 51]. This uncertainty in odometry can result in an erroneous speed model of the object to track.

4.2.2 Allocentric Modeling

Allocentric models are used to represent objects within an external frame of reference which is usually defined by landmarks. As mentioned earlier, the advantage in not using the perceiving robot but a different coordinate frame is the possibility to share the model with other robots which use the same reference system. Furthermore, in the RoboCup domain many objectives are easily stated in allocentric coordinates, e.g. *get to the goal*, *move back to a defensive position*.

To use an allocentric reference system for egocentrically perceived objects, robots determine their position within that frame of reference which is also known as *self-localization*. As soon as all robots have self-localized they can transform their locally perceived data into the allocentric coordinate system. Now the transformed data can be used by other robots of the group. The method of coordinate transformation by

using self-localization information is widely used within mobile robotics.

A transformation of an egocentric object position e, given the robot localization r into an allocentric position a can be performed in 2-d space as follows:

$$\begin{bmatrix} a_x \\ a_y \\ a_\theta \end{bmatrix} = \begin{bmatrix} \cos(r_\theta) & \sin(r_\theta) & 0 \\ -\sin(r_\theta) & \cos(r_\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} + \begin{bmatrix} r_x \\ r_y \\ r_\theta \end{bmatrix}, \tag{4.1}$$

where every variable e, r and a consists of an x and y component for the 2-d position and of θ for the orientation.

The back-transformation from allocentric in egocentric coordinates works accordingly:

$$\begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = \begin{bmatrix} \cos(r_\theta) & -\sin(r_\theta) & 0 \\ \sin(r_\theta) & \cos(r_\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \left(\begin{bmatrix} a_x \\ a_y \\ a_\theta \end{bmatrix} - \begin{bmatrix} r_x \\ r_y \\ r_\theta \end{bmatrix} \right) \tag{4.2}$$

Since self-localization can be erroneous, it would be useful to communicate sensory data without having to self-localize. Therefore, the next sections present an approach to using certain sensory data, that can be used for communication to other robots and for creation of allocentric models without self-localization necessary.

4.3 Shareable Sensory Data

A possibility for robots to communicate sensory data without being localized is to use multiple percepts within the same image, so called *percept-relations*, introduced in [41, 42]. When the object to localize was seen together with reference objects whose position is static and known, e.g., in soccer a flag (also called *beacon*) or a goal, the robot receives information about the distance between both objects. Percept examples within the Four-Legged League are given in Fig. 4.3. The position of both objects in relation to each other can be determined as presented in Fig. 4.6 (c). Percept-relations can be used to constrain the state space of possible object-relations. But most percepts do not contain sufficient information to calculate an exact position for an object, usually ambiguities remain. Therefore communication of multiple percept-relations between a group of robots is necessary to provide all agents with sufficient sensory data. The main difference of this approach compared to other approaches, e.g., the team-ball modeling approach in [87] is that the communication of percepts can be done before the robots create a model.

Percept-Correlations

In this section the sensory error correlation of different percepts within the same image, taken on the Aibo ERS-7, shall be analyzed. Therefore the sensory data standard deviations σ^l were measured, while the robot was taking images of the same scene: a flag, a goal, a line, a ball and combinations of these, i.e., a ball percept combined with all

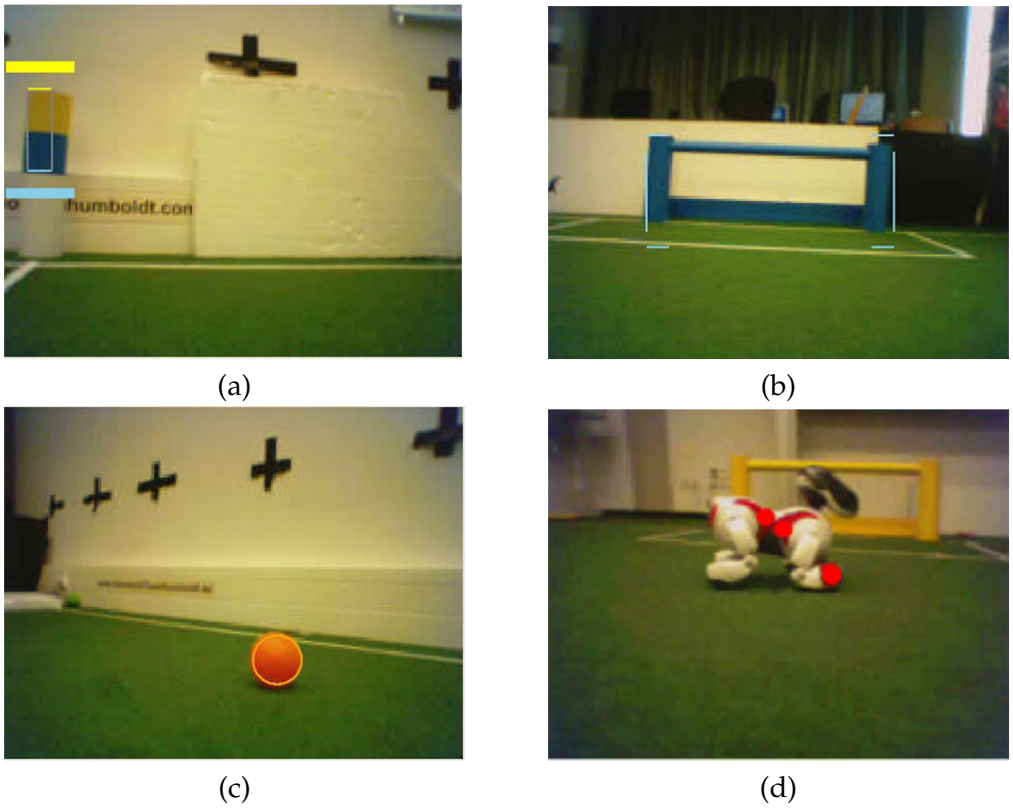


Figure 4.3: Percept examples within the Four-Legged League, 2007: (a) Flag percept, (b) goal percept, defined by the two goal posts, (c) ball percept, (d) robot percept, defined by the lowest colored point within the robot shape.

the other percepts. The perceiving robot was walking on the spot and keeping the distance to the object constant to perceive realistic image data, that was noised by walking motions.

We found out that the angle error of different percepts within the same image is strongly correlated. An example for a seen ball and a flag is given in Fig. 4.4. One possible reason for this is the coordinate transformation between the image plane and the egocentric coordinates (for example, joint angle sensory errors or unknown and unmeasured joint flexibilities). These errors would differ from frame to frame, but within a single frame should have an almost identical effect on the measured data. In the following section an implementation using percept-relations for object state modeling is described.

4.4 Object Modeling Using Percept-Relations

During a RoboCup game the robots scan their environment for different landmarks: goals, flags and the ball, as depicted in Fig. 4.5. We first introduce the domain, wherein

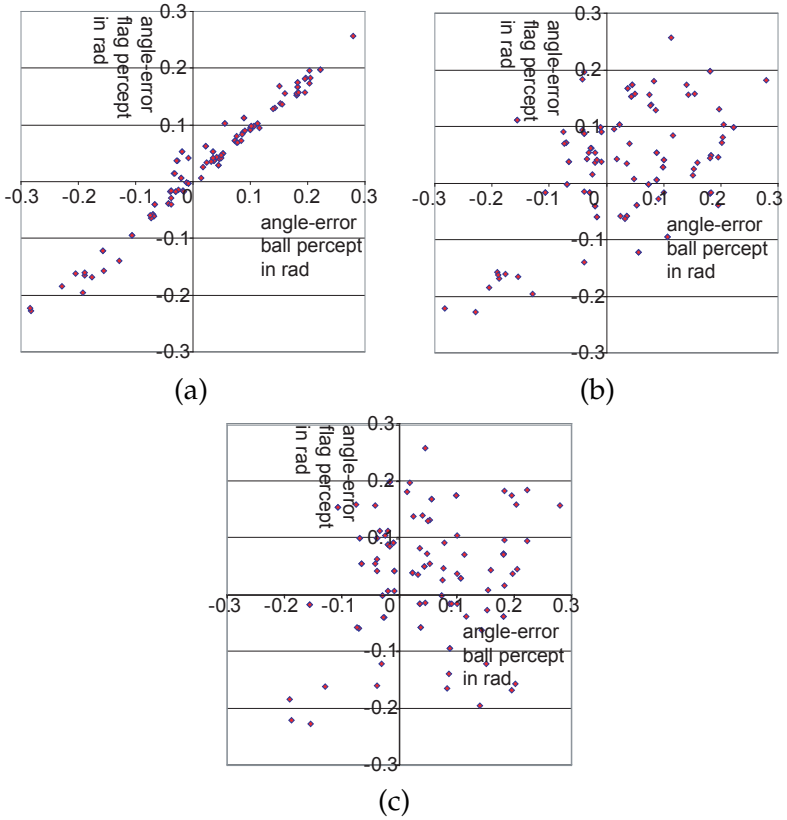


Figure 4.4: Percept error correlation over time while walking to a ball and a flag (recorded on the Aibo platform). Ball distance: 1.5m, flag distance: 2m. (a) Ball and flag within the same image - high error correlation; (b) ball was seen 0.03 seconds earlier than the flag, small error correlation; (c) ball was seen 0.2 seconds earlier than the flag - error correlation between both objects is almost zero.

the robots act and perceive images.

4.4.1 Information Gain by Single Percepts

When a robot perceives a two-color coded flag, it actually perceives the left and the right side of the flag and thus the angle between both sides. Together with the known size of the flag and the robots view height the robot can calculate the distance to the flag and the angle from which it perceives the flag, cf. Fig. 4.6 (a). In the here presented approach the information is not used for self-localization but for calculating the distance of objects, e.g., the flag and the ball to each other. When the robot is perceiving a goal, one possibility to generate a goal percept is to measure the angle between left and right goalpost. For a given goalpost angle, the robot can calculate its distance and angle to a hypothetical center point of a circle, where the circle lies on the two posts and on the

Figure 4.5: The soccer field of the Four-Legged League (2006).

robot camera position (Fig. 4.6 (b)). When a ball was seen, its position relative to the robot can be calculated. Lines or line crossings can be used as reference marks as well, as we will present later in this chapter.

4.4.2 Information Gain from Percept-Relations

When the object to localize was seen together with another landmark, e.g., a flag or a goal, the robot does not only get information about the distance to both objects but also information about the angle between both. With the *Cosine rule* one can calculate the distance between ball and landmark (Fig. 4.6 (c)).

When a goal and a ball were seen, a similar position estimation for the ball can be performed. The set of all remaining ball positions is described by a spiral arc, cf. Fig. 4.6 (d). With those examples shall be demonstrated, how visual percept-relations can be used to constrain the state space of all possible object positions. Most single percept-relations do not lead to unique solutions for object positions. Usually infinite solutions remain. A possibility to overcome this limitation is to search for further percept-relations, which is time consuming, because the robot has to turn its camera or even itself within the environment. Another possibility for an accurate estimation of object positions is to interchange percept-relations within a group of robots. This has two advantages:

- Despite the communication time, the information exchange between robots is relatively cheap in resources, because only few sensory data have to be transferred for the here described method.

- A group of robots can gather more sensory data than a single robot, because a group has access to more sensors and is usually better distributed within its environment.

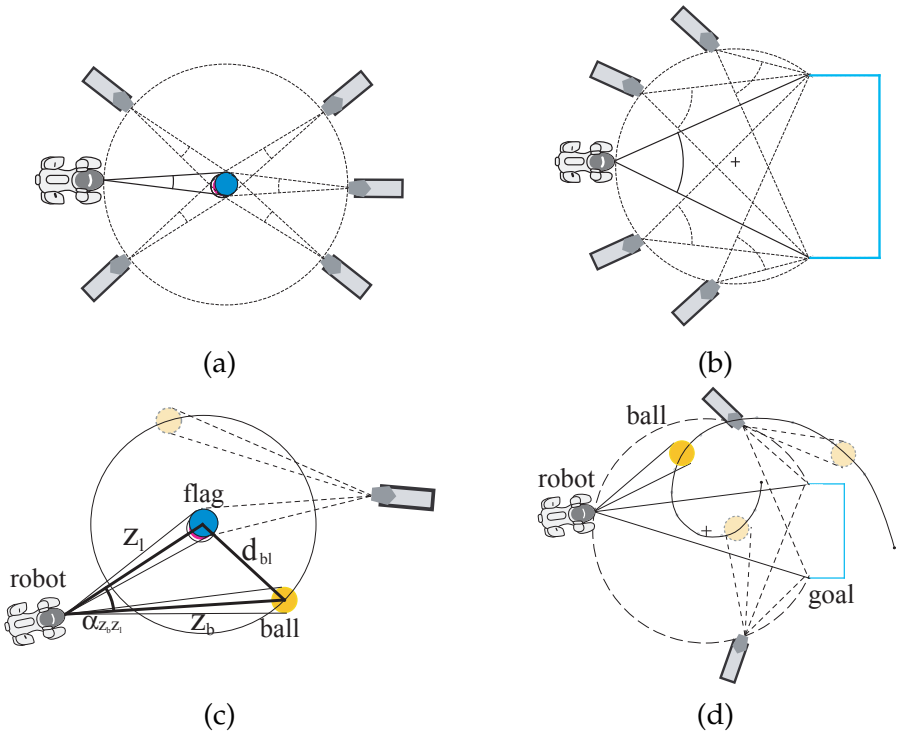


Figure 4.6: Sensor model derivation from percept-relations in RoboCup. (a) Flag perceived, robot calculates its distance. A circle, containing all possible positions, remains, (b) goal perceived (the angle between both goal posts), a robot can calculate its distance to a circle center point of a periphery circle. Two percepts within an image: (c) Flag and a ball, the robot determines the position of a ball in relation to a flag d_{bl}. (d) Goal and a ball, the spiral arc represents all possible ball positions.

In Fig. 4.7 one can see a two-agent scenario, in which two robots perceive the ball and different landmarks. Two circular and spiral arcs are calculated by the robots, to represent the ball position relative to the landmarks. When both agents communicate, two intersections between the circular and spiral arcs emerge. Thereby the number of possible ball positions is reduced to one or two points. Generally the set of remaining positions strongly depends on the sensor model, which depends on the properties of the landmarks. The less ambiguous a landmark is, the smaller is usually the solution space for all remaining object positions or self-localizing positions. After the introduction of the concept of percept-relations, an implementation of an object modeling approach, used on the Aibo ERS-7 is introduced. Percept data from the ERS-7 can be very noisy and can also lead to multimodal distributions, which has to be considered for the modeling process [16]. Therefore, in the presented approach a Monte-Carlo particle filter was used. Monte-Carlo Localization methods (MCL), have proven their power in numerous robot navigation tasks, e.g., in office environments [32], in

Figure 4.7: Experimental setup, two Aibos facing a ball and landmarks.

Percentage of Percept Occurances in Images			
Ball	Flag	Goal	Line
35	52	22	59
Only Ball	Ball and Flag	Ball and Goal	Ball and Line
3	24	8	28

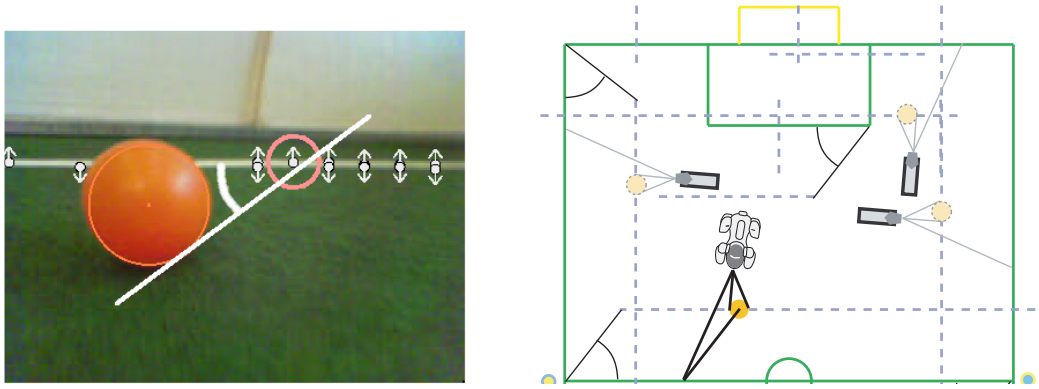
Table 4.1: This table gives an example of how many percepts and percept-relations were available during an certain amount of time.

the museum tour guide Minerva [113, 115, 24], outdoor applications in less structured environments where often map building is necessary as well [82, 88], and even for underwater navigation [64]. MCL is widely used in RoboCup for object tracking and self-localization [97, 73, 74, 112]. Other approaches as Multi-Hypotheses Tracking, grid-based approaches, Rao-Blackwellized particle filters [70] or geometric calculations [71] could be used as well as was demonstrated in the comparison of localization methods from Gutmann et al. [48, 49].

4.5 Ambiguous Landmarks

The sensory data introduced in the previous section was unique, i.e., it could be associated with objects, that existed only once within the robot's environment. Using a map, the robot can determine which object belongs to which sensory data. Usually there are ambiguous objects within a robot's environment as well, e.g., in the RoboCup domain there are many ambiguous field lines.

This line information can be a useful feature to reason about the robot's position [97] or about object positions. As can be seen in Table 4.1, field lines are very often present



(a) (b)

Figure 4.8: Percept- relations, using field line information. (a) A point of a line (small circle) was perceived with a ball. The ball distance to the line with respect to the line orientation is calculated. (b) Gray dotted lines represent all possible ball positions on the field, robot shapes represent possible positions of the robot.

in robot images and often together with other percepts as flags, goals or the ball. Now will be analyzed which information line data can bring. As an example a ball and a line are seen simultaneously.

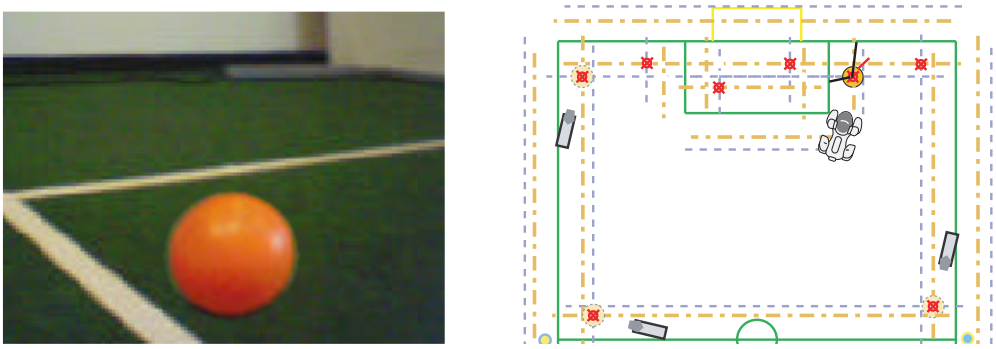
When the robot perceives a line, in our approach it actually perceives one or more points of the line, together with the normal vector of the line, as Fig. 4.8 (a) presents. When a ball is seen in the same image, the robot can calculate a shape, containing all possible ball positions on the field. When, e.g., a ball is seen 10 cm away from a line point in an angle of 45°, all points on the field are possible ball positions, which are in 10 cm distance and in a 45° angle to a line point on the field, cf. Fig. 4.8 (b).

In cases of more than one line percept it is interesting to know, in which relation the different percepts are [86]. Thereby, every line information can be used to improve the object localization or self-localization. If it is possible to put an object in relation to multiple lines, this strongly reduces the solution space for possible object positions.

Every ball-line-percept [40, 43] enables the robot to create a sensor model for possible ball positions on the soccer field, as presented in Fig. 4.8 (b). If two or more ball-line percept-relations $p(z_1|x), \ldots, p(z_n|x)$ are visible, the resulting sensor model can be calculated as the product of all ball-line percept-relation sensor models $p(z_i|x)$:

$$p(z_1, \ldots, z_n|x) = \prod_{i=1}^{n} p(z_i|x) \tag{4.3}$$

Just to mention, if the spatial relations of ball-line percept-relations are considered regarding the angles between different line percepts, then the resulting set for possible ball positions can further be decreased, as demonstrated in Fig. 4.9 (b). The following section will describe how an MCL, using percept-relations was implemented.



(a) (b)

Figure 4.9: Percept-relations using L-crossings, example. (a) The robot perceives a ball next to two different lines, (b) assuming the robot perceives points of two different lines (light gray and strong orange dotted in the image), the ball positions sensory model can be calculated as the product of the two ball-line percept-relation sensor models. Some possible ball positions can be excluded (depicted by crosses), when the angle between both lines is considered as well.

4.6 Monte-Carlo Localization - Implementation

A two-dimensional state space has been used for the ball tracking algorithm, to estimate the ball position on the field. For reasons of clearness we will use the symbol π to denote a particle. Every particle can be written as a state vector $\pi^{(i)}$:

$$\pi^{(i)} = \begin{bmatrix} q^i_{x_t} \\ q^i_{y_t} \end{bmatrix} \tag{4.4}$$

and its likelihood $\omega^{(i)}$. The likelihood of a particle $\pi^{(i)}$ is the normalized product of all likelihoods for all sensory data [97]. This means that the likelihoods for all landmark-ball combinations have to be calculated. For all given sensory data, e.g., a landmark l and a ball (with its distance and angle to the robot) the remaining possible ball positions relative to landmark l are calculated, as described in Section 4.4.2. The remaining curve of all possible ball positions is denoted by ξ^l.

It was described in 4.4.2 that ξ^l has a circular form, when l is a point landmark, e.g., a flag and a spiral form, when l is a goal, where only the angle between two posts was measured. The shortest distance δ^l from each particle $\pi^{(i)}$ to ξ^l is our argument for a Gaussian likelihood function $\mathcal{N}(\circ, \preceq, \supset)$, with $\mu = 0$ and with a standard deviation σ. The sensor model is assumed to be Gaussian which turned out to be a good approximation in experiments. The likelihoods are calculated for all seen landmarks l, then

		Standard Deviation σ	
Object	Distance in mm	σ_{Dst} in mm	σ_{Ang} in Rad
Ball	1500	170	0.015
Flag	2000	273	0.019
Goal	2000	25	0.021
Flag-Ball-Diff.	500	196	0.008
Goal-Ball-Diff.	500	175	0.0054

Table 4.2: Percept error correlations.

multiplied with each other and normalized by factor η:

$$\pi^{(i)} = \eta \prod_{l \in L'} \mathcal{N}(\delta^l, 0, \sigma) \tag{4.5}$$

In cases without new evidence all particles have the same likelihood. After likelihood calculation, particles are resampled. The estimated object position is calculated using a grid-based clustering approach.

4.6.1 Multi Agent Modeling

To incorporate the information from other robots, percept-relations are communicated between the robots. The receiving robot uses the communicated percepts for likelihood calculation of each particle the same way as if it was its own sensory data. This turned out to be very efficient:

- Some approaches communicate particle distribution by using n-trees, which can be useful when many objects are modeled in parallel or if the object position is ambiguous. In [87] this was done for cooperatively modeling the position of the ball by a group of Aibo-robots. But when two robots only know the arcs or the circular function on which the ball could be found and combining them, the entropy of the particle distribution not necessarily decreases.
- By communicating percept-relations rather than particle distributions, every robot can incorporate the communicated sensory data to calculate the particle distribution. Thereby one achieves a kind of sensor fusion rather than belief fusion as in case when particle distributions are communicated.

So, in the presented implementation every robot communicates every percept-relation (e.g., flag, ball) to other robots, to let every robot calculate the particle weights.

Sensor Model. For creating the sensor models, the standard deviation σ^l was measured by letting a robot take multiple images of different scenes: a ball, a flag, a goal and combinations of it. The standard deviation of distances and angles between objects in the image were measured as well. The robot was walking the whole time on the spot again, to get realistically noised sensory data. The experimental results are summarized in Table 4.2.

Algorithm 3: Ball tracking particle filter algorithm using percept-relations

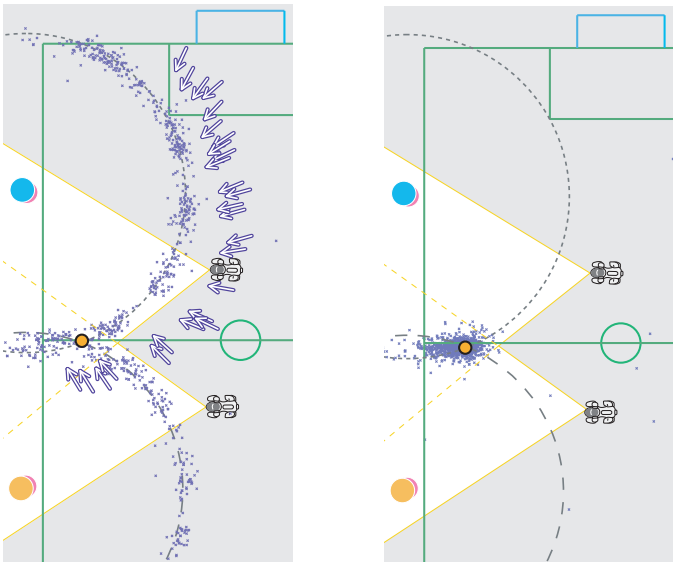
Input: $S_{t-1} = \{\langle \pi_{t-1}^{(i)}, \omega_{t-1}^{(i)} \rangle | 1, \ldots, n\}$ representing belief $Bel(x_{t-1})$, observation set Z_t

$S_t := \emptyset, \alpha := 0$ *// Initialization*
for $i := 1$ *to* n **do**
  Sample an index j from the discrete distribution given by the weights in S_{t-1}
  Sample $\pi_t^{(i)}$ from $p(x_t|x_{t-1})$ conditioned by $\pi_{t-1}^{(j)}$ (no control actions assumed)
  // Resampling: Draw state from previous belief
  $\omega_t^{(i)} := \prod_r \prod_l p(z_t^{l,r}|\pi_t^{(i)})$ *// Compute importance weight for all*
  // acquired percept relations l of all robots r
  $\alpha := \alpha + \omega_t^{(i)}$ *// Update normalization factor*
  $S_t := S_t \cup \{\langle \pi_t^{(i)}, \omega_t^{(i)} \rangle\}$ *// Insert sample into sample set*
end
for $i := 1$ *to* n **do**
  $\omega_t^{(i)} := \omega_t^{(i)} / \alpha$ *// Normalize weights*
end
return S_t

We found out that the standard deviation for the ball-flag distance (or ball-goal distance) is smaller than the sum of the distance errors given a single ball and a single flag (or goal). The same is true for the standard deviation of the angle. This indicates that the sensory error for percepts within the same image, caused by walking motions and head swings, is correlated. The idea of using local correlations for localization and map building has been used earlier for SLAM in [37, 89] and for object localization in [4, 17]. Because the here described experimental scenarios were static, we can abstract from network communication time, i.e., the time it takes to communicate percepts from one robot to the other. The algorithm for ball tracking by using percept-relations is presented in Alg. 3.

4.6.2 Self-Localization

For self-localization, the algorithm described in [97] was applied. A three dimensional state space was used, two dimensions for the field position of the robot and one dimension for its orientation. As sensory update data served the angle to the goal posts and to the flag boundaries as in [97], plus in our approach the distance and angle to the modeled ball which was modeled in allocentric coordinates. Experimental results are described within the next section.



(a) (b)

Figure 4.10: Cooperative ball localization with two flags, Exp. A: (a) No percept-relations communicated, robots are self-localizing (arrows represent SL-particles of the upper robot schematically), ball positions (cloud of dots) are modeled egocentrically, then transformed into global coordinates. Globally modeled ball particle distribution is then communicated to the other robot and combined with its ball particle distribution (union of particle sets). (b) Robots communicating percept-relations for calculating the particle distribution for the ball; the small circle at the center line marks the real ball position in the given experiment.

4.7 Experimental Results

The Aibo ERS-7 robot served as a test platform. We compare our algorithm to an approach where robots communicate particle distributions. The algorithm works as follows: Two robots tried to localize and to model the ball in an egocentric coordinates. As a result each robot maintained a particle distribution for possible ball positions, resulting from self-localization belief and the locally modeled ball positions. Without communication neither robot was able to accurately determine the ball position (Experiment A,B). Now the two robots communicated their particle distributions to each other. After communication each robot created a new particle cloud as a combination of its own belief (the own particle distribution) and the communicated belief (communicated particle distribution). This algorithm was compared to the here presented algorithm in situations, where self-localization is not possible, e.g., when every robot can only see one landmark and the ball. The following experiments were performed, all experiments included two robots:

1. Experiment A: Each robot perceives a flag and the ball. The ball position is modeled.
2. Experiment B: One robot perceives a goal and a ball, the other one perceives a flag and a ball. The ball position is modeled.
3. Experiment C: Same setup as in experiment A, now one robot uses the modeled ball position for self-localization.
4. Experiment D: One robot perceives a goal and a ball, the other one a line and a ball. The ball position is modeled.
5. Experiment E: Same setup as in experiment D, instead of a goal, one robot perceives a flag. The ball position is modeled.

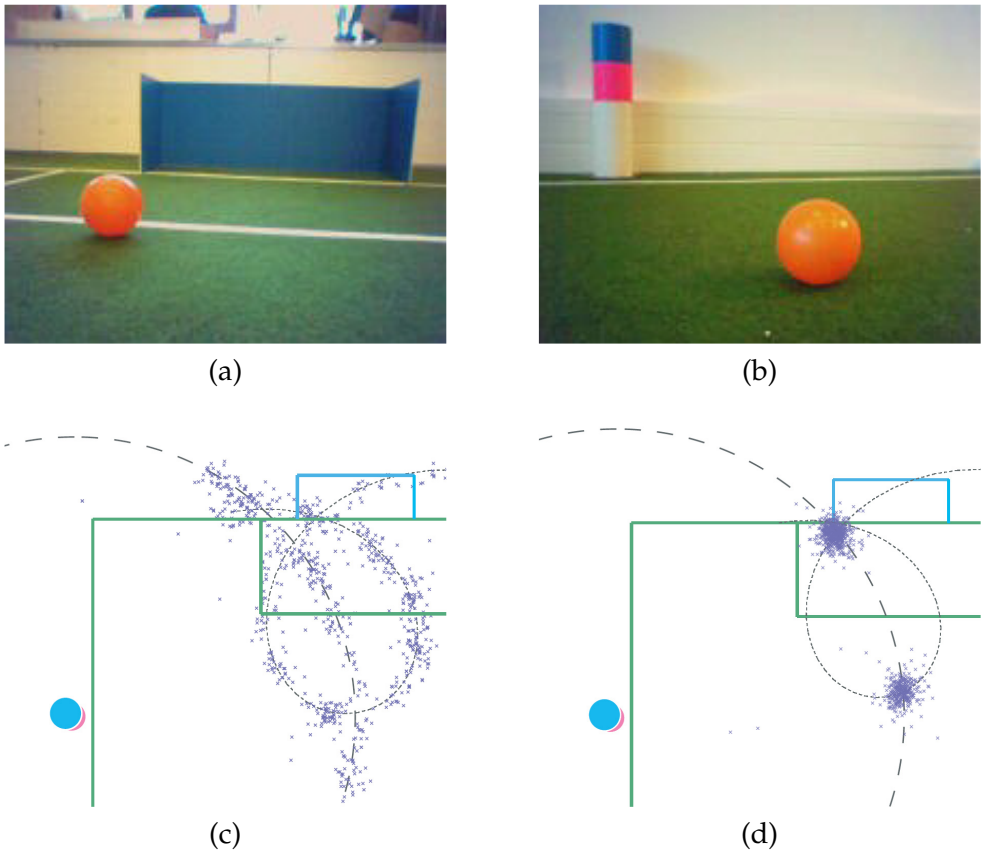


Figure 4.11: Cooperative ball localization with a flag and a goal, Exp. B: (a) One robot perceives a goal, (b) another robots sees a flag; (c) resulting ball position particle distribution of both robots without communication, as in Fig. 4.10 (a). In (d) two robots are communicating percept-relations.

Non-Ambiguous Sensory Data. In the first experiment, both robots were placed in front of different landmarks with partially overlapping fields of view, so that both robots could see the ball (Fig. 4.10). Experiments indicated that there is no conver-

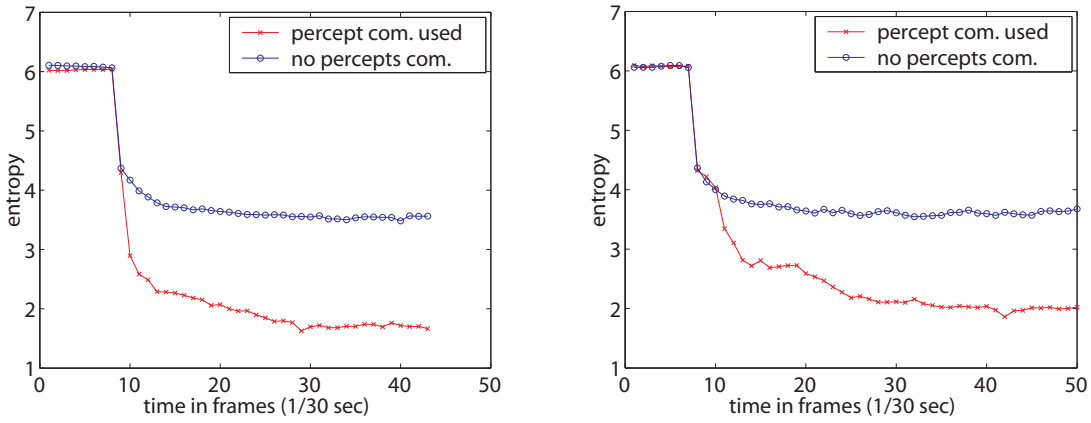
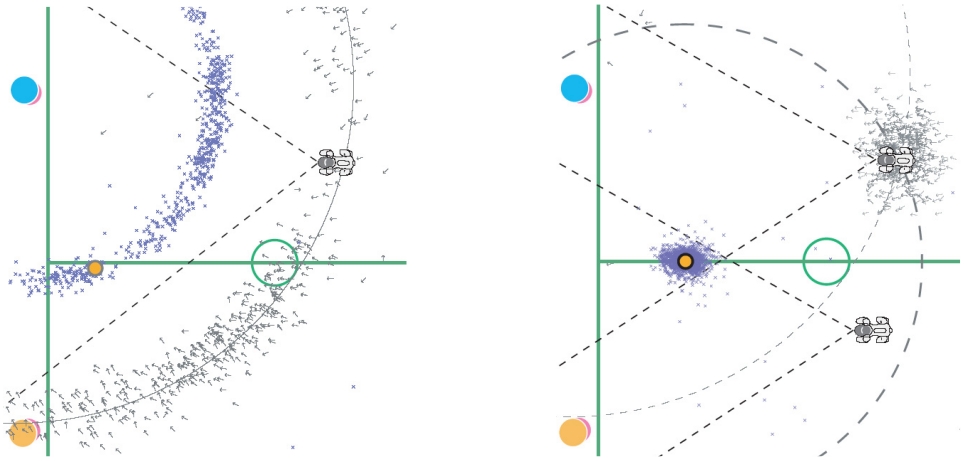


Figure 4.12: Entropy comparison of cooperative ball position modeling. Percept-relations not communicated (blue curve) vs. communication of percept-relations (red curve). 100 samples, 240 grid cells (2-d), cell size 10 cm x 10 cm. (a) Exp. A, two seen flags: using object relations leads to a lower entropy. (b) Exp. B, one goal and one flag were seen: the entropy is much lower when using object relations. The particle cloud converged within a fraction of a second.

gence to a confined area in case when the two robots are communicating their particle distributions to each other and combine their distributions. When percept communication is applied, the particle distribution converges to a confined area. The entropy of the particle distribution confirms this quantitatively. As presented in Fig. 4.12 (a), the entropy is decreasing slightly because the particle distribution converges circular to the flags, but not to a small area. The entropy decrease is much stronger in case where percept-relations are communicated, cf. Fig. 4.12 (a). An intersection of the two particle distributions can lead to the same solution as for communicating percept-relations, but usually requires more data to be communicated. Furthermore, one has to define how the intersection of two particle distributions is calculated, e.g., through discretization of the particle sets by grids. In the second experiment, one robot was placed in a way that it could see a flag and the ball, the other one in front of a goal and a ball (Fig. 4.11 (a,b)). Again the robots tried to self-localize and communicated their particle distributions. The result was compared to the algorithm, that communicated percept-relations. As in experiment A, without communication no convergence of particles occurred. The particle distribution was the result of the spiral shaped distribution generated by the robot seeing the goal and the ball, combined with the circular distribution of the robot seeing the flag and the ball.



(a) (b)

Figure 4.13: Cooperative ball and self-localization using two flags, Exp. C: (a) One robot is perceiving the ball and self-localizing by the upper flag. A circular particle distribution remains for the robot positions (bigger circle) and the ball positions (smaller circle). (b) Two robots localizing the ball with percept-relations, the upper robot is localizing, using its distance to the upper flag and its distance to the modeled ball position. Two particle clouds can be seen, one for the ball, one for the robot.

Applying the algorithm where percept-relations were communicated, two confined particle areas remained for the ball position estimation. Again, the entropy was decreasing more in case of percept-relation communication compared to particle distribution communication, cf. 4.12 (b). The entropy for two seen flags (experiment A) remained lower than for a seen goal and a flag (experiment B), because the second possible ball position was in case A outside the field. In Fig. 4.12 one can see that the particle distribution converged quickly. In the next experiment one robot was put in front of a flag and a ball. Again it had to self-localize. The used reference algorithm was the self-localization approach described in [97]. As the robot could only see one landmark, the particle distribution did not converge to a certain area, two circular clouds remained, one for the ball and one for the self-localization particle distribution, cf. Fig. 4.13 (a). An accurate self-localization was impossible. Again, when the two robots were not interchanging percept-relations, the ball particle distribution did not converge, as presented in Fig. 4.10 (a). Now the two robots had to cooperatively determine the ball position using percept-relations, each robot could use its own distance and angle to the ball for the likelihood calculation of the localization particle set. Fig. 4.13 (b) depicts how self-localization can be improved when using percept-relations and the modeled position from the allocentric ball model. The lower entropy of the self-localization particle distribution proves quantitatively that using position data from objects that were modeled in allocentric coordinates can reduce the self-localization ambiguity, as Fig. 4.14 shows.

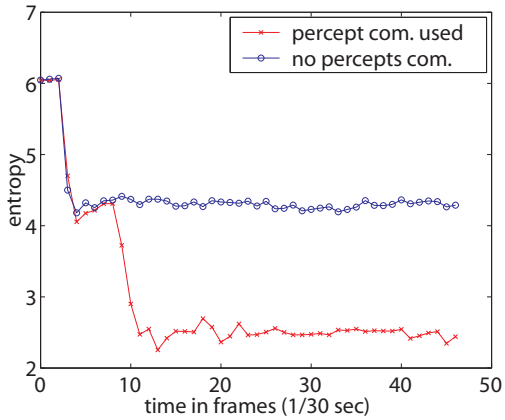


Figure 4.14: Entropy comparison of cooperative self-localization, Exp. C: The upper blue function represents the self-localization entropy when no percept-relations were used, the lower red function the entropy for communicated percept-relations. Entropy decreases when the robot perceives the flag but remains at a high level; The self-localization entropy becomes much lower when using visual percept-relations for ball modeling.

Ambiguous Sensory Data. In the next experiment, two robots try to localize and to model the ball egocentrically again. Now line-based percept-relations are applied. Each robot has a particle distribution for possible ball positions in allocentric coordinates, resulting from self-localization belief and the locally modeled ball position. In the given situation neither robot is able to accurately determine the ball position on its own. Later both robots communicate their particle distribution to each other. After communication, each robot creates a new particle cloud as a combination of its own belief and the communicated belief (communicated particle distribution). Now it was checked, how this algorithm performs in contrast to the percept-relation algorithm in situations, where self-localization is not possible, e.g., when every robot can only see one landmark and the ball.

In the experimental setup, both robots were placed in front of different landmarks, one in front of a goal and one in front of a line with partially overlapping fields of view, in such a way that both robots could see the ball, cf. Fig. 4.15 and 4.16. The robots could accurately model the ball position when just communicating particle distributions, whereas by communicating percept-relations the modeled position converged to two small areas (Fig. 4.16). The entropy measurement proves this quantitatively in Fig. 4.18 (a), the entropy was much smaller, when percept-relations were communicated.

In Experiment E (Fig. 4.17) one robot perceived a flag, the other robot perceived a line and both could see the ball. Again the robots tried to self-localize and to model the egocentric ball position. Then they transformed the egocentrically modeled ball particles into allocentric coordinates and communicated the particle distribution to each other. Simple particle communication lead to a relatively weak convergence of the resulting

(a) (b)

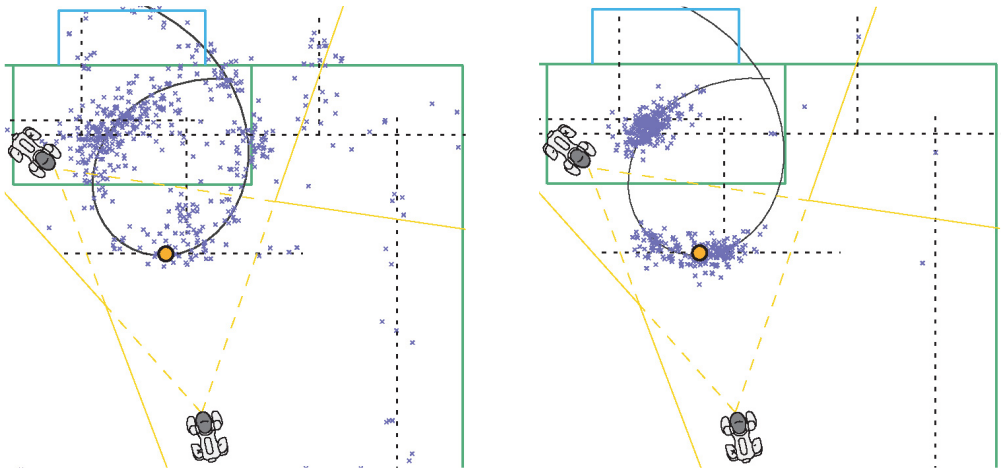
Figure 4.15: Images with ball-line percept-relations: (a) Robot A seeing a ball and a goal, (b) robot B seeing a ball and a line.

particle distribution. The particle convergence was higher when percept-relations were communicated, cf. Fig. 4.17 (b) and the entropy became smaller, cf. Fig. 4.18 (b).

4.8 Conclusions

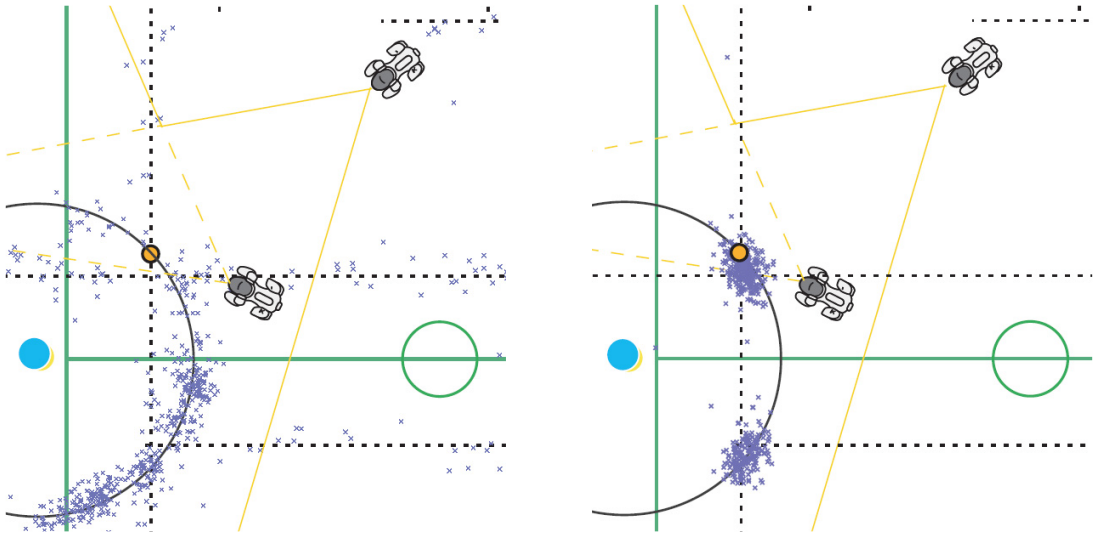
Object-relations in robot images can be used to localize objects in allocentric coordinates, e.g., if a ball is detected in an image next to a goal, the robot can infer something about where the ball is on the field. Without having to be localized at all, the robot can accurately estimate the position of an object within a map of its environment using only percept-relations. Furthermore, it could be demonstrated how the process of object localization can be sped up by communicating percept-relations to other robots. Two non-localized robots were able to localize an object using their sensory input in conjunction with communicated percept-relations. In a next step was presented how the gained knowledge about allocentric object positions can be used for an improved Markov self-localization.

Altogether, it depends on the situation and sensory data availability, which sensory data to use. Hybrid algorithms, using percept-relations to improve classic models are an option as well. So far the presented algorithm could be used for static object positions. The next chapter analyzes how the concept of percept-relations can be used to model moving objects.



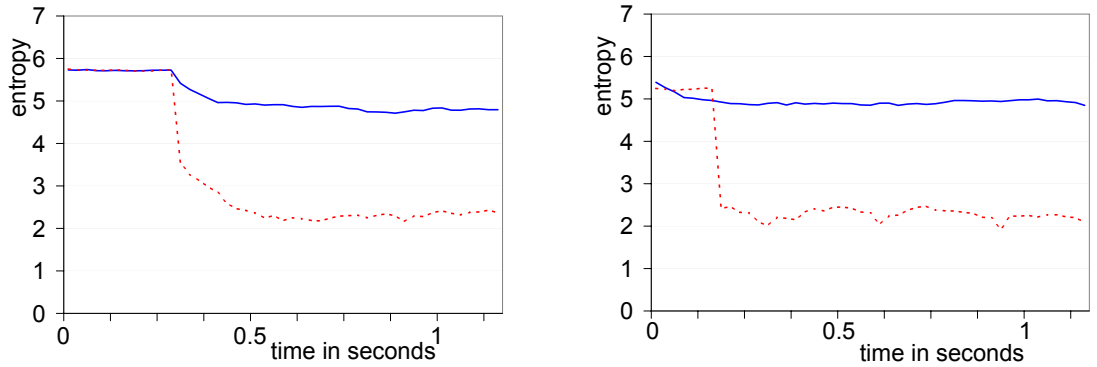
(a) (b)

Figure 4.16: Cooperative ball localization using goal and line data, Exp. D: (a) Both robots try to localize and they have an egocentric ball model. After interchanging their particle distribution, the particle cloud does not convergence to a confined area.
(b) Robots interchange the percept-relations (ball-line and ball-goal), then they are updating and resampling their particle distributions. The presented distribution converges quickly to two small areas.



(a) (b)

Figure 4.17: Cooperative ball localization using flag and line data, Exp. E. The upper robot can see the ball and a flag, the lower robot can see the line and the flag. (a) Communicating particles does not lead to a convergence of the particles. (b) Communicating percept-relations leads to convergence of the particle cloud into two small areas.



(a) (b)

Figure 4.18: Entropy comparison of cooperative ball position modeling using line data: The red curve represents the particle entropy when communicating percept-relations, the blue curve represents the particle entropy when not communicating percept-relations. (a) Entropies for Experiment D. (b) Entropies for Experiment E.

5 Cooperative Dynamics Modeling and Communication

The following approach demonstrates, how a group of agents can cooperatively model the speed of an object without being localized. The idea is to use percept-relations as input data for a non-linear regression function and to see, if this can improve the speed modeling accuracy. The following results were created in collaboration with Alexander Block, who wrote his Diploma theses under the supervision of this work's author.

5.1 Modeling of Dynamics Using Percept-Relations

Within the next sections, parameters describing different ball trajectories are derived. As depicted in Fig. 5.1, given the trajectory of the ball, the distance of the ball to the flag $dist_i$ at time t_i can be calculated for every time step. Then t_l is the time of the closest distance of the ball to the flag. $dist_l$ is the corresponding closest distance to the flag. Given the ball has a constant speed v, the length of the hypothenuse $dist_i$ can be calculated using *Pythagoras' law*. The catheti of the triangle are the closest distance of the ball trajectory to the landmark $dist_l$ and the distance $d_{i,l}$ between ball position at time t_i and t_l. We have $d_{i,l} = v(t_i - t_l)$. Now the distance of the ball to the flag can be calculated by:

$$\mathrm{dist_i} = \sqrt{[v(t_i - t_l)]^2 + \mathrm{dist_l}^2} \tag{5.1}$$

Since the percept-relations of ball and flag provide distances between those two, given $(dist_i, t_i)$ at least two percept-relations are necessary to calculate a constant ball speed trajectory and at least three percept-relations to calculate a decreasing ball speed trajectory.

To calculate the velocity for constant ball speeds, the constant speed equation (5.1) was transformed into a polynomial form, in order to simplify calculation. We get:

$$\mathrm{dist_i}^2 \overset{for v \neq 0}{=} v^2 {t_i}^2 + (-2v^2 t_l) t_i + \frac{(-2v^2 t_l)^2 + 4v^2 \mathrm{dist_l}^2}{4v^2} \tag{5.2}$$

This equation has the form $y = ax^2 + bx + c$, where $y = \mathrm{dist_i}^2$ and $x = t_i$. The parame-

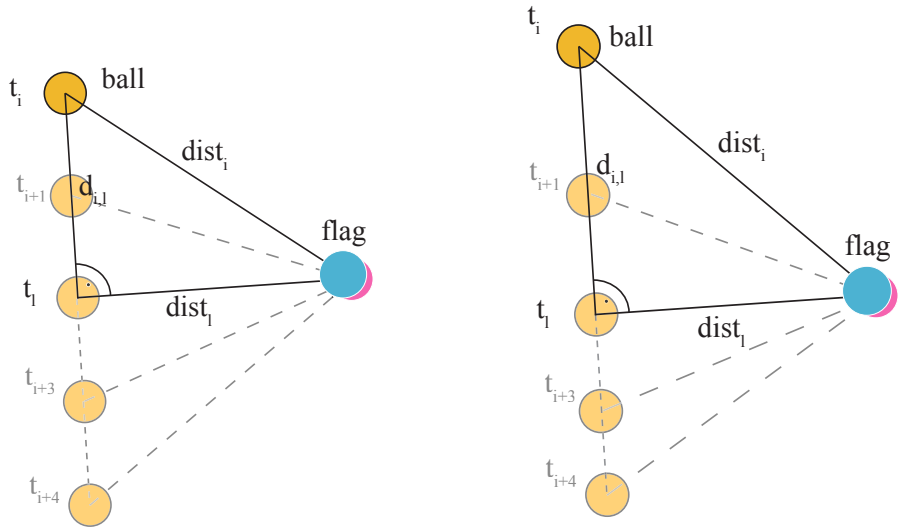


Figure 5.1: Ball trace while passing a flag, constant and decreasing speed: A triangle is constructed from the closest point of the ball trajectory to the landmark, the current ball position at time t_i and the landmark position. Using *Pythagoras' law* the current distance $dist_i$ of the ball to the landmark can be calculated. (a) depicts the ball trajectory over time, given a constant ball speed, (b) for decreasing speed.

ters to estimate are:

$$a = v^2 \tag{5.3}$$

$$b = -2v^2 t_l = -2at_l \tag{5.4}$$

$$c = \frac{(-2v^2 t_l)^2 + 4v^2 \mathrm{dist_l}^2}{4v^2} = \frac{b^2 + 4a\mathrm{dist_l}^2}{4a} \tag{5.5}$$

Using linear regression, one can solve the equation and derive the parameters $\hat{a}$, $\hat{b}$ and $\hat{c}$. With $x = t_i$ and $y = \mathrm{dist_i}^2$ we get for $\hat{a}$:

$$\hat{a} = \frac{\overline{x^2y}\,\overline{x^2} - \overline{x^2y}\,\overline{x}\,\overline{x} - \overline{xy}\,\overline{x^3} + \overline{x}\,\overline{y}\,\overline{x^3} - \overline{y}\,\overline{x^2}\,\overline{x^2} + \overline{xy}\,\overline{x}\,\overline{x^2}}{\overline{x^2}\,\overline{x^4} - \overline{x}\,\overline{x}\,\overline{x^4} - \overline{x^3}\,\overline{x^3} + 2\overline{x}\,\overline{x^2}\,\overline{x^3} - \overline{x^2}\,\overline{x^2}\,\overline{x^2}}, \tag{5.6}$$

where $\overline{x}$ is the average of all x and $\overline{xy}$ is the average of all $x_i y_i$. The linear regression provides for $\hat{b}$ an estimation:

$$\hat{b} = \frac{\overline{xy} - \hat{a}\overline{x^3} - \overline{x}\,\overline{y} + \hat{a}\overline{x}\,\overline{x^2}}{\overline{x^2} - \overline{x}\,\overline{x}} \tag{5.7}$$

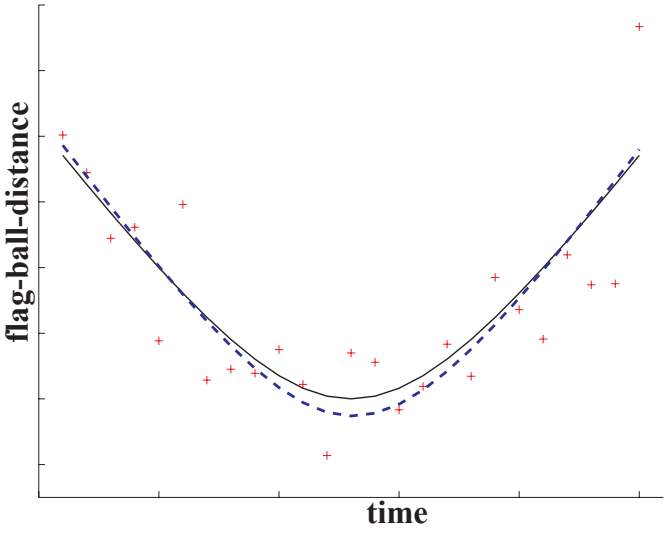


Figure 5.2: Moving ball, distance to flag over time, constant speed: The dashed blue curve represents the estimated hyperbola, generated by 25 measurements (red crosses). The black curve describes the real motion of the ball.

The solution for $\hat{c}$ is easy to comprehend:

$$\hat{c} = \overline{y} - \hat{a}\overline{x^2} - \hat{b}\overline{x} \tag{5.8}$$

From the final estimations $\hat{a}$, $\hat{b}$ and $\hat{c}$ the resulting parameters of equation (5.1) can be found. We have

$$\hat{v} = \sqrt{\hat{a}}$$
$$\hat{t_l} = \frac{-\hat{b}}{2\hat{a}}$$
$$\hat{\text{dist}_l} = \sqrt{\frac{4\hat{a}\hat{c} - \hat{b}^2}{4\hat{a}}}$$

Here $\hat{v}$ denotes the estimated ball speed and $\hat{dist_l}$ the estimated distance of the ball to the flag. Those coefficients minimize the sum of squared differences of the observed measurement values for the regression equation. Fig. 5.2 illustrates an hyperbola generated from 25 measurement values. As can be seen, the estimated hyperbola approximates the real function quite well, although sensory data are very noisy.

5.1.1 Friction Modeling

When a ball is rolling across the field, its speed is not constant but decreasing until the ball finally stops. Although it is possible to generate friction models offline for a given ball and carpet and then to use them during modeling, it is also possible to calculate

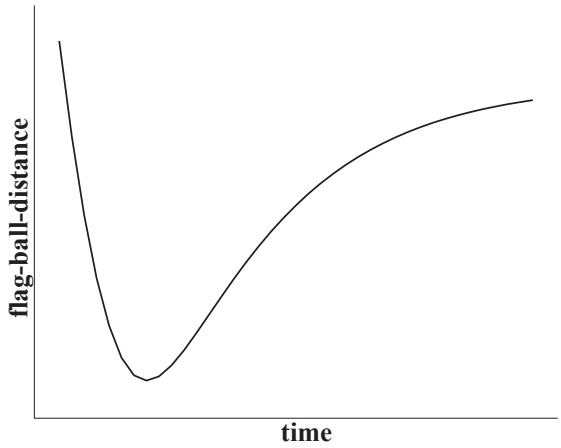


Figure 5.3: Moving ball, distance to flag over time, decreasing speed. The distance function converges to the final distance, where the ball stops.

the decreasing speed coefficient from the sensory data. Fig. 5.1 (b) presents a ball passing a flag with decreasing speed over time, which is indicated by the decreasing ball position differences within constant time steps. Experiments indicated that the coherence between time and ball position can be approximated by an exponential function (Fig. 5.3). The function converges to a distance value at which the ball finally stops. To create a discrete model it was assumed that the speed of the ball is decreasing by a constant factor $\delta = (1 - \text{friction-coefficient})$ in each time step: $v_t = v_{t-1}\delta$. The distance, the ball is moving during a single time frame is given by $s_i = v_i \cdot \Delta t$

Thus, the moved distance in time interval $[t_i, t_{i+1}]$ is δ-fold as long as the distance in time interval $[t_{i-1}, t_i]$ and so on. Summing up the partial distances from time step to time step, the distance from t_i till t_l can be derived:

$$\begin{aligned} d_{i,l} &= s_i + s_{i+1} + \cdots + s_{l-1} \\ &= v_l\delta^{(i-l)}\triangle t + v_l\delta^{(i-l)+1}\triangle t + \cdots + v_l\delta^{-1}\triangle t \\ &= \delta^{(i-l)}\triangle t \sum_{j=0}^{(l-i)-1} v_l\delta^j. \end{aligned} \tag{5.9}$$

The partial sums are geometric series $s_n = \sum_{k=0}^{n} a_0 q^k$ and can be calculated by:

$$s_n = a_0 \frac{1 - q^{(n+1)}}{1 - q}, \tag{5.10}$$

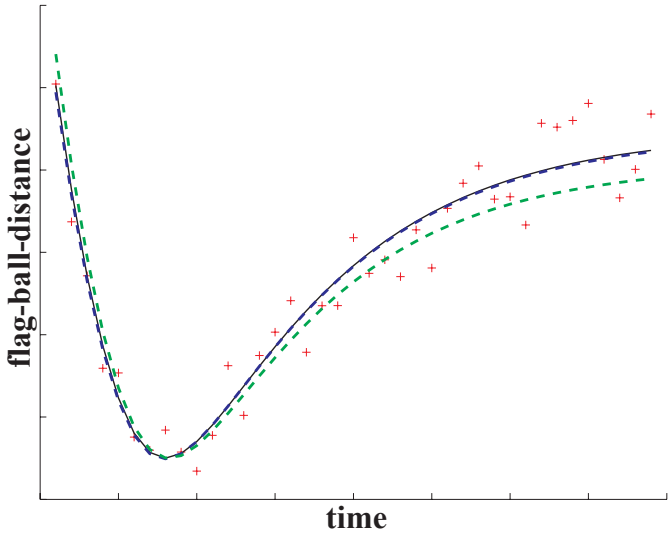


Figure 5.4: Estimating the decreasing ball speed over time. Depicted are 30 measurements $(t_i, dist_i)$ (red crosses), the ground truth ball distance trajectory (solid black), the estimations for a given degression factor (dashed blue) and unknown degression factor (dashed green) δ.

Equation (5.9) can be transformed into

$$\begin{aligned} d_{i,l} &= \delta^{(i-l)} v_l \left(\frac{1-\delta^{-(i-l)}}{1-\delta} \right) \triangle t \\ &= \left(\frac{v_l}{1-\delta} \right) \delta^{(i-l)} (1-\delta^{-(i-l)}) \triangle t \\ &= \left(\frac{v_l}{1-\delta} \right) (\delta^{(i-l)} - 1) \triangle t. \end{aligned} \tag{5.11}$$

Now one can calculate the distance from the ball to the landmark in every time step t_i by:

$$\mathrm{dist_i} = \sqrt{\left(\frac{v_l}{1-\delta} \right)^2 (\delta^{(i-l)} - 1)^2 + \mathrm{dist_l}^2}. \tag{5.12}$$

If the parameters of the motion trajectory have to be calculated, one can use non-linear regression. Here the Gauss-Newton approach was used for estimation of the parameters t_l, v_l, $dist_l$ and δ from 39 measurement pairs.

Fig. 5.4 presents the results of the non-linear regression. The dashed green function is an approximation of ground truth (black) when δ is not given. The blue curve describes the estimation when δ is given. Experiments showed that for known degression factors the speed estimation is very accurate. Also for unknown degression factors the function is a good estimation to the ground truth at the beginning. However, for bigger

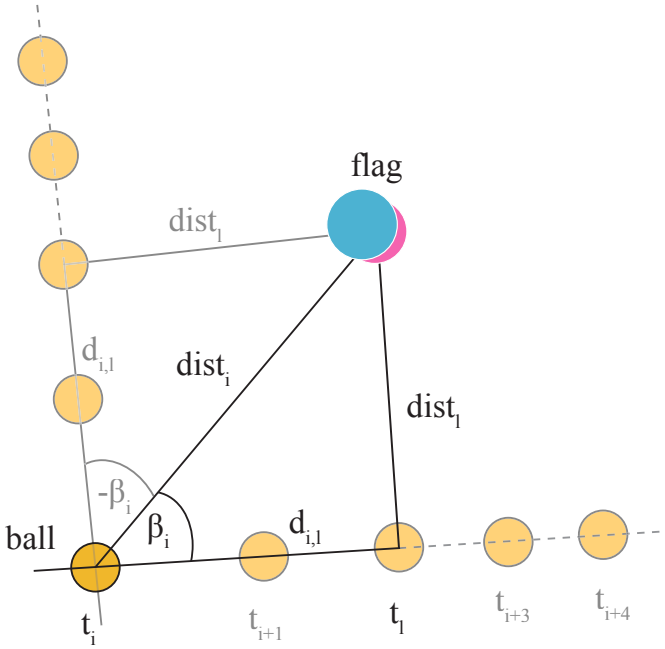


Figure 5.5: Resulting ball trajectories relative to flag. With the distance based approach and a given position two possible trajectories remain: $\beta_i^{(1)} = \beta_i$ and $\beta_i^{(2)} = -\beta_i$.

time values both functions diverge. It was described how the degression factor can be calculated online from sensory data, but for efficiency reasons it is useful to generate this factor once and then to assume that it stays constant, because object frictions are usually not changing over time in most domains like, e.g., in RoboCup. After having modeled the trajectory parameters, the current speed v_i can be calculated using

$$v_i = v_l \delta^{(i-l)}. \tag{5.13}$$

Obviously, with the given approach it is only possible to calculate the speed amount on the field but not its motion direction on the field. This limitation can be overcome as soon as this model is combined with the position of the ball on the ground. As presented in Fig. 5.5, one can calculate the two possible trajectories for any given starting point.

5.1.2 Experimental Results

In experiments the ball rolled down a ramp and moved at constant speed across the field. The perceiving robot, an Aibo ERS-7 was tracking it and perceiving a flag at the same time. Fig. 5.6 summarizes the results of this experiment. One can see the point of smallest ball-to-flag distance and the decreasing ascent of the curve, that indicates the decreasing ball speed on the field.

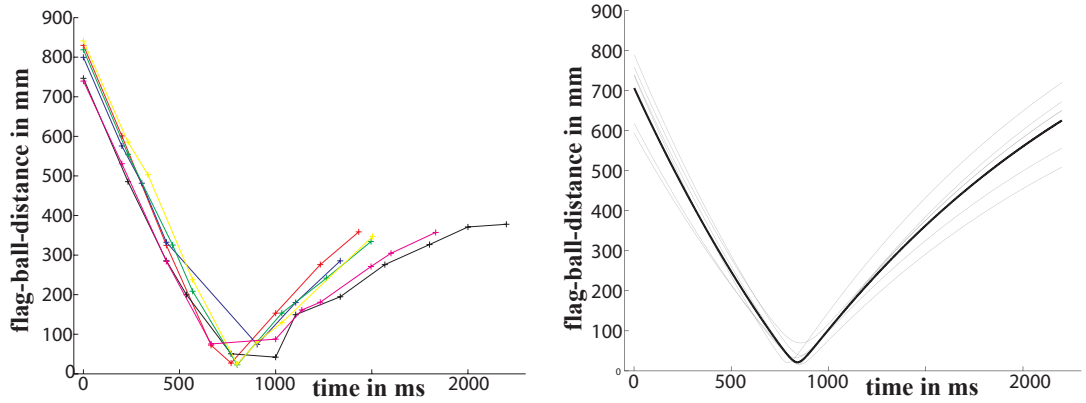


Figure 5.6: Ball trajectory estimation experiments. (a) Ball distance-time measurement of 6 runs, raw data. (b) Visualization of the estimated distance function using non-linear regression. The black curve depicts the average function.

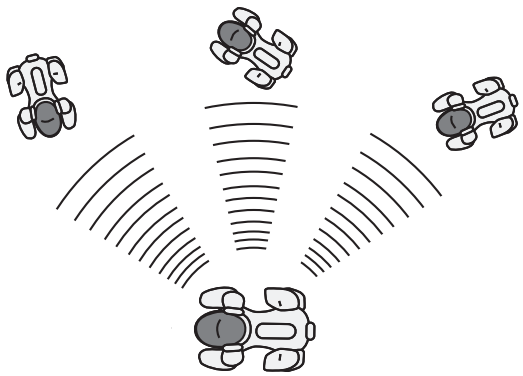


Figure 5.7: Robot communication, broadcasting sketch. A robot broadcasts the perceived sensory data to all the other robots.

5.2 Dynamics and Time Components

In this section is demonstrated, how sensory data of different robots can be used when there is a considerable time gap between sending and receiving communication data. In dynamic scenarios it is important to consider at which time the sensory data was perceived. This task becomes more complex whenever communication delays are not constant but changing over time, which happens often during communicating over Wireless LAN, e.g., on the Aibos.

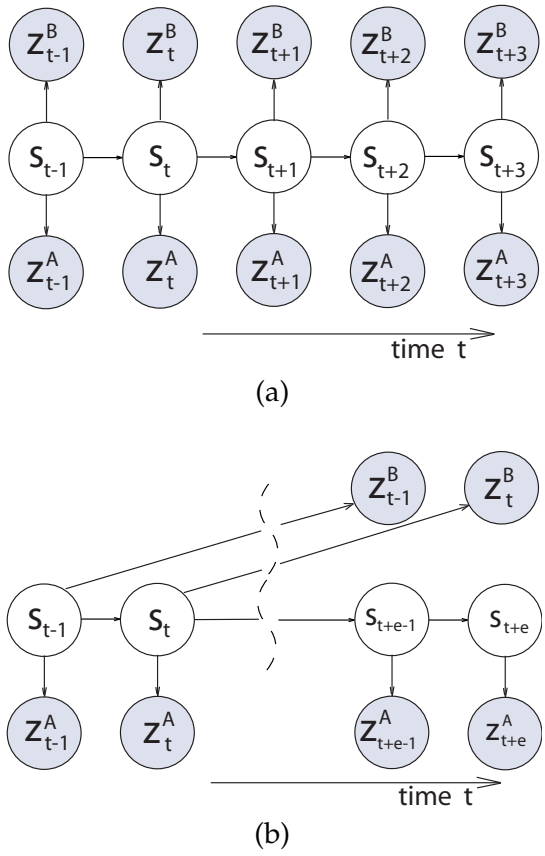


Figure 5.8: Markov model, no communication delay and constant delay. Model for robot R_A. Robot R_B is communicating its percepts. (a) No communication delay. (b) Constant communication delay e, $e \gg 0$.

5.2.1 Synchronization of Communication

Time synchronization of agents within a network can be achieved to a limited extent only, because time delays are changing permanently. However, this synchronization is very important. It is usually achieved through NTP-protocols, as in the GermanTeam. The synchronized time was called *team time*. Without going further into detail, NTP-protocolls measure the round-trip times of messages through the network, i.e., the time it takes for the message to get from sender to the receiver through different layers and back. After the robots agreed on a team time, this time can be assigned to every percept the robots send to each other. Now every robot can calculate how old the percept is whenever a percept was received over the network. In the next scenario robots use their own sensory data and communicated sensory data for modeling (cf. Fig. 5.7).

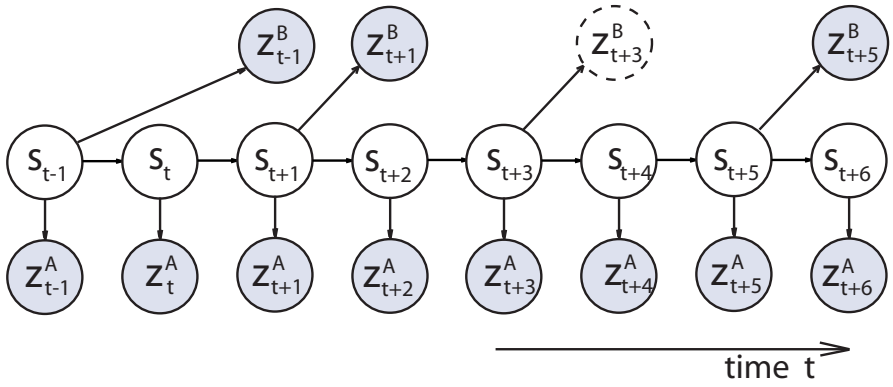


Figure 5.9: Altering communication delay. Communication frequency is lower than perception frequency. Communication delay can change over time, and communication frames can get lost, implementation example from GT 2007 [98].

5.2.2 Handling Communication Delays

In the next section is demonstrated how communication delays can be handled. Hidden Markov models estimate the current object state using the last state estimation plus the executed actions and current sensory data, as described earlier. In case of delayed communication, the robot has to wait for the communicated sensory data, or it can incorporate its own sensory data only. Fig. 5.8 illustrates, how constantly delayed sensory data affect the state estimation accuracy. In real situations the communication delay changes over time and even worse, frames can get lost, cf. Fig. 5.9.

Fig. 5.10 presents round-trip times measured on an Aibo which was communicating with another Aibo. Code was used from the GermanTeam 2007 [98]. Furthermore, the data confirmed that the robots were communicating sensory data within each fourth cognition step only, i.e., they received images four times as often as they sent and received data to and from other robots.

Now we want to give an example, where modeling errors can affect communication delays, when they are not treated correctly. Imagine two robots tracking a moving object in relation to two reference objects. The task is to model the position of the tracked object. Each robot is perceiving the moving object in relation to another landmark. The object is moving with constant speed and direction. As already demonstrated, the robots can estimate the ball position accurately as long as a map is given and as long as the situation remains static. But if the ball is moving and if there is a delayed communication this can be fatal for the object position estimation, as Fig. 5.11 presents. The communicated sensory data leads then to wrongly modeled ball positions resulting in a non-linear trajectory estimation.

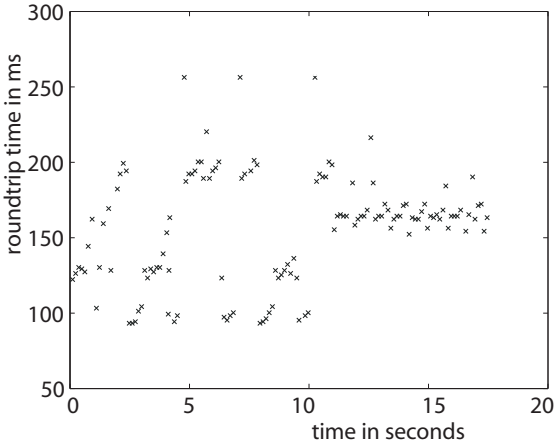


Figure 5.10: Communication round-trip time. The time between sending a percept and receiving it is about half of the round-trip time.

5.2.3 Future Prediction, History Revision

Now a simple approach is introduced, to handle communication delays, which is called *future prediction, history revision*. It is an extension to the usual Bayes filter approach. The main difference is that not just one state is predicted and later corrected but many. Given a robot R_0 modeling object state s_t at time t and with communication delay $e > 0$. Furthermore, n robots $R_1, \ldots, R_n$ are communicating. The robot R_0 has to wait until time frame $t + e$ to get all sensory data of communicating robots $R_1, \ldots, R_n$. Now it is assumed that all robots are observing the object of interest at time $t + e + 1$. If state s_{t+e+1} shall be estimated, s_{t+1} has to be estimated at first, because at time $t + e + 1$ all communicated sensory information is available for state s_{t+1}. The a-posteriori state estimation $\hat{s}_{t+1}$ for s_{t+1} is stored, because it is the most current belief which was generated from on sensory data of *all* robots. The following states s_{t+2}, s_{t+3},..., s_{t+e+1} have to be estimated a-priori, i.e., by using sensory and motion data from robot R_0 alone, because the communicated sensory data for those states is still not available. Due to the fact that the robot has to make a prediction for s^-_{t+e+1} with partial information, starting from state $\hat{s}_{t+1}$, this process is called *future prediction*. The problem is that this prediction usually needs to calculate the estimates $\hat{s}^-_{t+2}$ till $\hat{s}^-_{t+e}$ as well.

Thus, this model can be interpreted as an e-th order Markov chain depending on the communication delay e. At time $t + e + 2$ the sensory data $z^{R_1}_{t+2}, \ldots, z^{R_n}_{t+2}$ from the other robots $R_1, \ldots, R_n$, which was recorded at time $t + 2$ is received. The a-priori state estimation $\hat{s}^-_{t+2}$ can now be revised to the a-posteriori $\hat{s}_{t+2}$, based on the additional communicated information. To perform the revision, the last a-posteriori state estimation $\hat{s}_{t+1}$ was stored earlier and is restored now. Modeling continues, resulting in an a-priori estimation $\hat{s}^-_{t+e+3}$. Therefore, because state estimate s^-_{t+2} changed, all state estimates between s^-_{t+2} and s^-_{t+e+3} have to be revised as well. Thus, this approach is only applicable in cases where time delays are small. Otherwise the revision effort for all a-priori mod-

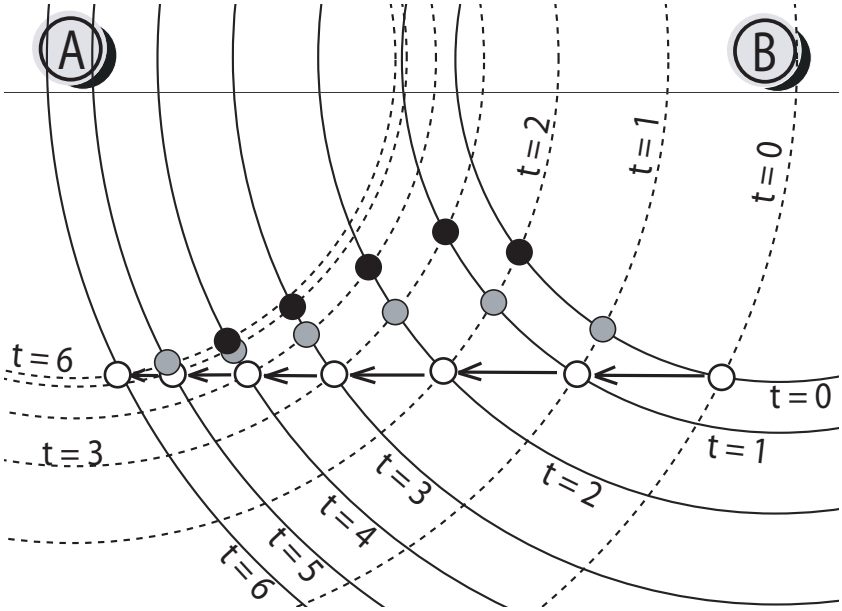


Figure 5.11: Possible impacts of communication delay on cooperative ball trajectory modeling. The white ball is moving from right to left. Robot R_A (not visible) is perceiving the ball and flag A, robot R_B is communicating its percept-relations, resulting from the ball and flag B. Traces depict modeled ball positions from robot R_A's view. Gray balls represent the modeled ball positions when communication delay is one time step. The black ball trace represents the effect, when communication is delayed by two time steps.

eled states can become very big.

5.3 Distributed Calculation

Different alternatives to handle the computation of the cooperative model exist. One approach is to have a robot handling all the sensory data, creating the model and communicate the state estimation to all the other robots. The advantage is that computational resources of just one, or at least a subgroup of the robots is necessary. But every approach has to handle situations in which the calculating robot breaks down. The following alternatives are considered:

- One robot does all the calculation, e.g., the one with the lowest ID, see Fig 5.12 (a).
  Pro: The resources of the other robots remain unaffected. If the calculating robot breaks down, this can be recognized by missing answers of that robot and the next robot jumps in for him.
  Contra: This approach can become quite complicated. When communication is error-prone, a delayed communication could be interpreted as a failure of the

calculating robot. The negotiation about which active robot is next in line can be an unwanted overhead.

- A small group of robots is doing the calculation, see Fig 5.12 (b).
  Pro: If one robot fails the chance is high that other robots are remaining, providing a model.
  Contra: If multiple robots are modeling the same state, the question which of the models should be taken by the other agents remains. Again, the robots could agree on always taking the model of the robot with the smallest ID.

- Every robot maintains its own model, see Fig 5.12 (c).
  Pro: The approach is highly robust to breakdowns of other robots.
  Contra: Every robot has to maintain a model, which requires a certain amount of calculation resources.

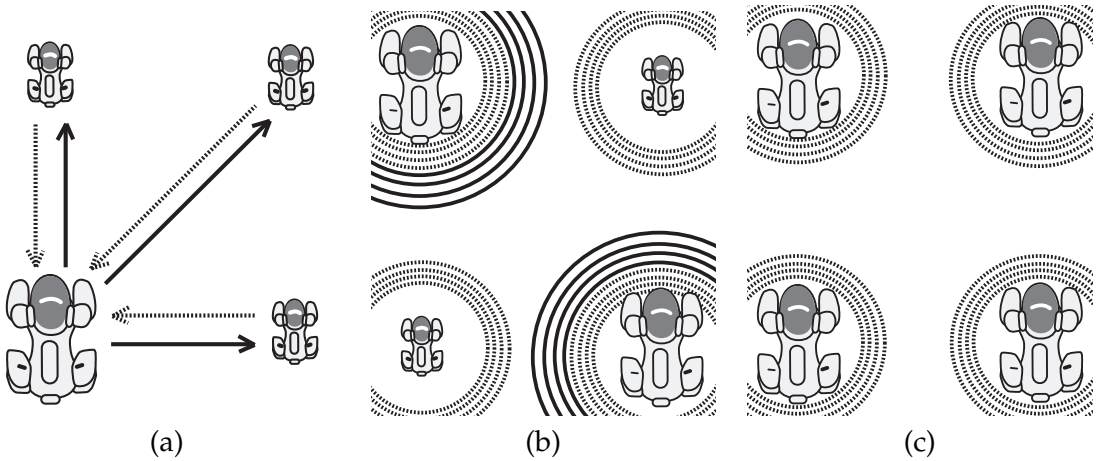


Figure 5.12: Calculation distribution examples: (a) One robot perceives all sensory data, creates the state model and returns it to all the other agents. (b) A subset of robots, here two robots, is perceiving sensory data and modeling and communicating the state to the other robots. (c) All robots are modeling the state. Small circles represent broadcasted percept data, big circles represent state model broadcasts.

5.4 Conclusions

We presented a method, that enables a group of robots to estimate the velocity of moving objects, e.g., a ball by using reference objects. This approach can be used to enhance classic speed modeling approaches, that are using self-localization information and robot motion data. Furthermore, we demonstrated how delays in the transmission of sensory data between robots can lead to incorrect object models. It was described

how small and constant communication delays can be treated by an n-th order Markov chain, but in general the handling of altering communication delays is hard. The last section of this chapter focussed on different variants of distributed model calculation as a tradeoff between redundancies and computational resources. Within the RoboCup domain where wireless communication breakdowns are common, it turned out to be a good solution to have every robot maintaining its own model.

Summarizing, the last two chapters focussed on distribution functions of sensory data in static and dynamic environments and introduced some aspects of communication and distributed computing. The next chapters introduce an alternative approach to Bayesian modeling that can be used for single- and multi-agent world modeling and that can be applied to a robot in the RoboCup domain.

6 Theory of Constraint Based Modeling

In this chapter a localization technique, using constraints for belief and sensor modeling is introduced. The advantage of constraints is their ability to represent arbitrary belief distributions. Kalman filters and their extended forms EKF and UKF use Gaussians, which without extensions only can represent unimodal beliefs. Multi-Hypotheses Tracking can overcome this restriction to a certain degree, but still all hypotheses within a MHT only represent unimodal beliefs. Some belief functions, e.g., those described in [23] are hard to describe by a union of Gaussians. Particle filters do not have this limitation but they have to maintain a high number of samples to work accurately. Therefore, they cannot be used in high-dimensional spaces. Using a smaller number of particles can lead to good localization approximation results as well, as has been proven for the RoboCup Four-Legged League by Lenser et al. [73] and by Röfer and Jüngel [97], where small particle sets with a hundred samples where used.

Especially when computational power is limited, one has to come back to highly optimized approaches. When sensory data is ambiguous, e.g., when there are many solutions within the state space, that match the sensory data as described by Fox in [33], a higher number of particles is necessary. On robots with less processing power as the Aibo or many humanoid robots, the computational limitation becomes significant for localization approaches because image processing, planning and motion generation need a big percentage of the available resources.

Related Work. Constraint-based localization techniques have been applied earlier. In [111] constraints are used to generate two-dimensional Gaussian estimates for robot localization. A cooperative and constraint based object search strategy for a group of robots is described in [7]. Another localization method using bounded-error state estimation was introduced for a small truck or vehicle, which was equipped with ultrasonic sensors in [65, 108]. Methods applied within this work where interval mathematics [84, 57]. A paper, that describes how to compute sets that guaranty to contain all solutions of sets of nonlinear inequalities is given by Jaulin and Walter [56]. Within the next sections basic concepts of constraint based localization are presented, including the generation of constraints from sensory data, propagation of different constraints and handling of inconsistencies. Some of the concepts, especially within the theoretical part, were developed in collaboration with Hans-Dieter Burkhard, Kateryna Gerasymova and Heinrich Mellmann. Those concepts are used in the following chapters to describe implementations for constraint based localization approaches.

6.1 Fundamentals of Constraint Satisfaction Problems

A constraint C is defined over a set of variables V. The values every variable $v_i \in V$ can take is defined by its domain $Dom(v_i)$. A constraint C defines for a variable set which values the variables of V can take [46]:

$$C \subseteq \mathrm{Dom}(v_1) \times \cdots \times \mathrm{Dom}(v_k) \tag{6.1}$$

More formal, all constraints are defined over the set of all variables $v_1, v_2, \ldots, v_k$ but not all variables have to be affected by the constraint. The domain of a variable v is denoted by $\mathrm{Dom}(v)$, and the whole universe under consideration is given by

$$U = \mathrm{Dom}(v_1) \times \cdots \times \mathrm{Dom}(v_k). \tag{6.2}$$

In this work, all domains $Dom(v)$ are considered to be continuous intervals of real values, i.e., $U \subseteq \mathbb{R}^k$.

Definition 6.1.1 *(Constraints)*

1. *A **constraint** C over $v_1, \ldots, v_k$ is a subset $C \subseteq U$.*
2. *An assignment β of values to the variables $v_1, \ldots, v_k$, i.e. $\beta \in U$, is a **solution** of C iff $\beta \in C$.*

Definition 6.1.2 *(Constraint Sets)*

1. *A **c**onstraint set $\mathcal{C}$ over $v_1, \ldots, v_k$ is a finite set of constraints over those variables: $\mathcal{C} = \{C_1, \ldots, C_n\}$.*
2. *An assignment $\beta \in U$ is a **s**olution of $\mathcal{C}$ if β is a solution of all $C \in \mathcal{C}$, i.e. if $\beta \in \bigcap \mathcal{C}$*
3. *A constraint set $\mathcal{C}$ is **i**nconsistent if there is no solution, i.e. if $\bigcap \mathcal{C} = \emptyset$*

Applying these concepts to robotics, the perception and modeling of the environment can be defined using those constraints as was presented in [44]. Given a picture of a scene, there are constraints between objects in the image and objects in reality. Object parameters, image parameters and camera parameters are bound by constraints. Consecutive robot positions are in strong correlation with each other and can be described as constraints containing odometry, control data and velocities.

Furthermore, one has to deal with noisy and ambiguous sensory data. Ambiguous sensory data result in an ambiguous object state. Noisy sensory data can result in inconsistencies. For those paradigms we introduce quality measures.

There is one further important aspect in using constraint solving techniques for mobile robotics. In contrast to other CSPs, in mobile robotics there has to be a solution in reality for a set of constraints, even if those constraints are inconsistent with each other. To find such a solution, one can use certain algorithms which are introduced within this work and analyzed with regard to possible applications.

6.2 Generating Constraints from Sensory Data

This section describes, how sensory data can be represented by constraints. The focus lies on visual sensory data, i.e., sensory data from camera images and the resulting percepts. The robot can perceive different sensory data, some of the sensory data can be associated with a certain object, other sensory data is ambiguous as, for example, line data. Those aspects, e.g., sensory information type, if the object is moving or static, etc. result in different constraints.

6.2.1 Distance Based Constraints

If the Euclidean distance to an object is known and if the object has no or just a small dilation, e.g., like a flag on the soccer field, the resulting constraint can have a very simple structure.

The constraint has to contain information about the measured distance and measurement errors. The number of variables within the constraint depends on the dimension of the state space and on the object, that was perceived. For example, assume a robot perceived a unique point within distance r. Let σ be the standard deviation of the measurement error. The constraint about the x_R- and y_R-coordinates of the robot, given the landmark position x_L, y_L looks as follows:

$$C = \{(x_R, y_R) | (r-\sigma) \leq \sqrt{(x_R - x_L)^2 + (y_R - y_L)^2} \leq (r+\sigma)\} \tag{6.3}$$

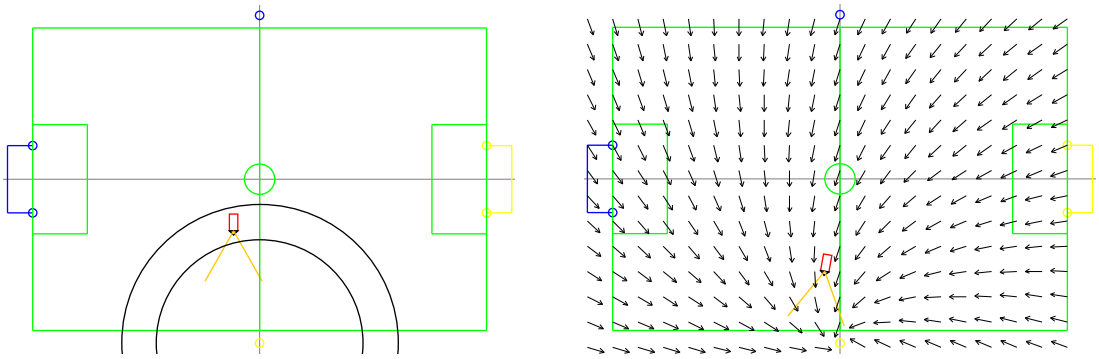
Fig. 6.1 (a) depicts the distance based constraint graphically.

6.2.2 Bearing Based Constraints

Bearing based measurements are useful whenever the distance to an object can not be determined directly, e.g., because the size of the object is unknown. In most cases the *bearing*, i.e., the angle to an object then still can be measured. To give an example, the distance to an airplane in the sky is hard to measure but the bearing, i.e., the angle to it can be measured. A constraint created from a bearing measure on a 2-d plane with a given bearing angle γ to a landmark looks as:

$$C = \{(x_R, y_R, \alpha_R) | \arctan \frac{y_L - y_R}{x_L - x_R} - \theta_R = \gamma\}, \tag{6.4}$$

where x_R, y_R, θ_R are the x, y coordinates of the robot on the field, θ_R its orientation, and x_L, y_L the coordinates of the landmark. Figures 6.1 (b) and 6.2 illustrate which position-angle combinations are defined by a bearing measurement constraint within the RoboCup domain. In this example, sensory errors were not considered. Some sensors can measure the distance r and the angle γ to certain landmarks. Then one can



(a) (b)

Figure 6.1: Constraint variants for different sensory data: (a) A measured distance to a landmark, (b) a measured bearing to a landmark.

generate constraints which give hint about the position and the orientation of the robot:

$$C = \{(x_R, y_R, \alpha_R) | \left| \begin{pmatrix} x_R \\ y_R \end{pmatrix} - \begin{pmatrix} x_L \\ y_L \end{pmatrix} \right| = r \wedge \arctan \frac{y_L - y_R}{x_L - x_R} - \alpha_R = \gamma\} \tag{6.5}$$

Further Constraint Examples. The so far described constraints were generated from uniquely identifiable objects, whose position is given by a map. However, there can be ambiguous objects as well. A robot moving through an office area usually does not know, which wall it is perceiving even when it possesses a map of the building in which it is moving. The robot could stand in front of every wall available when it is perceiving a wall within a certain distance. In the next section will be described how one can handle ambiguous sensory data using constraints.

6.3 Ambiguous Sensory Data

Ambiguous sensory data can be treated as a set of alternatives of perceived objects. This can be demonstrated by, e.g., a field line. When a robot is perceiving a point of a line, and if it can estimate the distance to that point, it can calculate its position constraint C_p as in Fig. 6.1 (a). Assuming not only one but a possibly infinite set of points $\mathbf{P} = \{p_i | p_i \in I\}$ can be found within the environment of a robot, then a constraint $C_{p'}$ can be derived, containing all possible robot positions for all possible seen points $\mathbf{P}$, as Fig. 6.3 depicts.

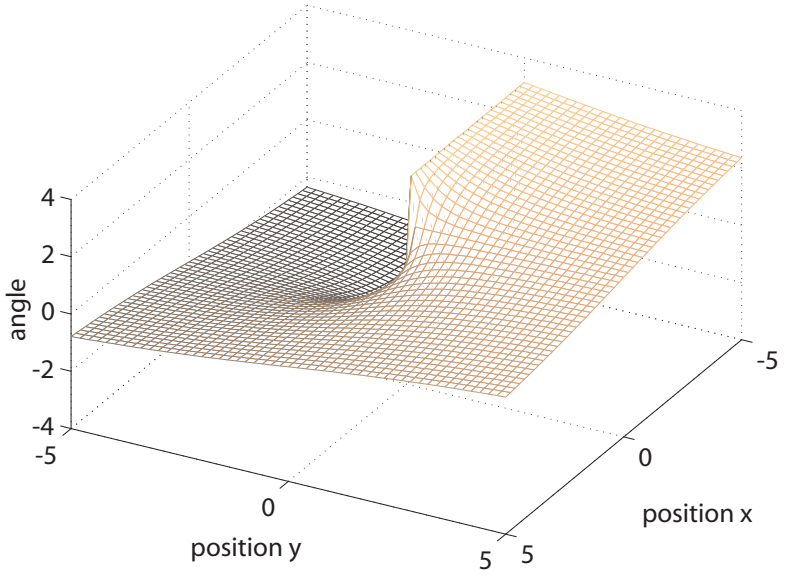


Figure 6.2: Sensor model for a bearing-only measurement, 2-d bearing measurement and the resulting possible robot positions x, y, θ.

6.4 Constraint Propagation

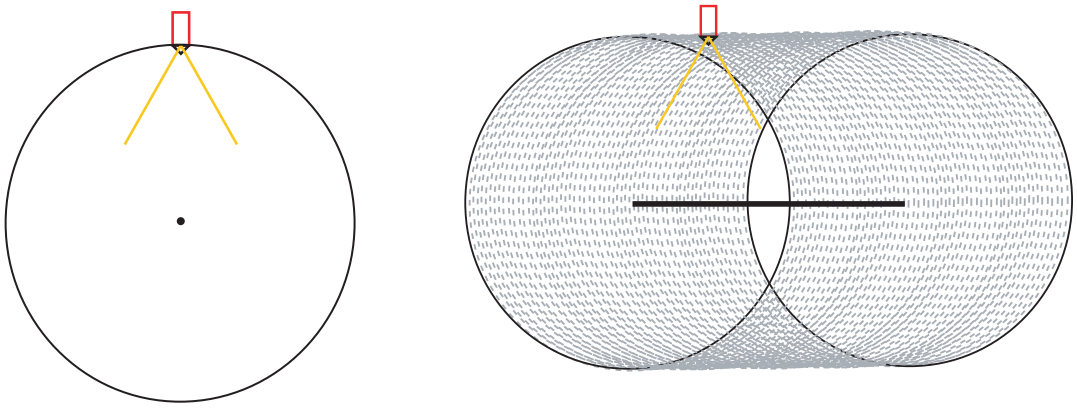
Known techniques for constraint problems as, e.g., described in [21, 47] produce successively reduced sets leading to a sequence of decreasing restrictions: $U = D_0 \supseteq D_1 \supseteq D_2, \supseteq \ldots$ Restrictions for numerical constraints are often considered in the form of k-dimensional intervals $I = [a, b] := \{x | a \leq x \leq b\}$ where $a, b \in U$ and the $\leq$-relation is defined componentwise. The set of all intervals in U is denoted by $\mathcal{I}$. A basic scheme for constraint propagation with:

- a constraint set $\mathcal{C} = \{C_1, \ldots, C_n\}$ over variables $v_1, \ldots, v_k$ with domain $U = \mathrm{Dom}(v_1) \times \cdots \times \mathrm{Dom}(v_k)$,
- a selection function $c : \mathbb{N} \rightarrow \mathcal{C}$ which selects a constraint C for processing in each step i,
- a propagation function $d : 2^U \times \mathcal{C} \rightarrow 2^U$ for constraint propagation which is monotonously decreasing in the first argument: $d(D, C) \subseteq D$,
- a stop function $t : \mathbb{N} \rightarrow \{true, false\}$,

works as follows:

Definition 6.4.1 *(Basic Scheme for Constraint Propagation, BSCP)*

- *Step(0) Initialization:* $D_0 := U$, $i := 1$
- *Step(i) Propagation:* $D_i := d(D_{i-1}, C_i)$.



(a) (b)

Figure 6.3: Ambiguous sensory data, resulting sensor models: (a) A robot perceives a unique feature which can only be found once within the environment, resulting positions depicted as a circle, (b) if the robot perceives a point which belongs to a line, its position can be on every point within the gray area.

- *If* $t_i = true$*: Stop.*
- *Otherwise* $i := i + 1$*, continue with Step(i).*

Any algorithm which is defined according to this scheme is called a *BSCP-algorithm*. The restrictions are used to shrink the search space for possible solutions. If the shrinkage is too strong, possible solutions may be lost. For that, backtracking is allowed in related algorithms.

A basic selection strategy is a round robin over all constraints from $\mathcal{C}$, while more elaborate algorithms use some heuristics. A more sophisticated stop criterion t considers the changes in the sets D_i. Note that the sequence needs not to become stationary if only $D_i = D_{i-1}$. Actually, the sequence $D_0, D_1, D_2, \ldots$ needs not to become stationary at all.

For localization problems with simple constraints, e.g., hyper-intervals it is possible to compute the solution directly:

Corollary 6.4.1 *If the propagation function d is defined by $d(D, C) := D \cap C$ for all $D \subseteq U$ and all $C \in \mathcal{C}$, then the sequence becomes stationary after $n = \mathrm{card}(\mathcal{C})$ steps with the correct result $D_n = \bigcap \mathcal{C}$.*

For simpler calculations, the restrictions D_i are often taken in simpler forms, e.g., as intervals and the restriction function d is defined accordingly.

Usually constraint satisfaction problems need only some but not necessarily all solutions. For that, the restriction function d does not need to regard all possible solutions, i.e., it does not need to be conservative according to the definition 6.4.2. A commonly used condition is local consistency:

Definition 6.4.2 *(Locally consistent propagation function)*

1. *A restriction D is called* **l***ocally consistent w.r.t. a constraint C if*

$$\forall d = [d_1, \ldots, d_k] \in D \quad \forall i = 1, \ldots, k \quad \exists d' = [d'_1, \ldots, d'_k] \in D \cap C \; : \; d_i = d'_i$$

   i.e., if each value of a variable of an assignment from D can be completed to an assignment in D which satisfies C.

2. *A propagation function $d : 2^U \times \mathcal{C} \rightarrow 2^U$ is* **l***ocally consistent if it holds for all D, C: $d(D, C)$ is locally consistent for C.*

3. *The* **m***aximal locally consistent propagation function $d_{\max lc} : 2^U \times \mathcal{C} \rightarrow 2^U$ is defined by $d_{\max lc}(D, C) := \max\{d(D, C) | d \text{ is locally cons.}\}$.*

Since the search for solutions is easier in a more restricted search space – as provided by smaller restrictions D_i – constraint propagation is usually not performed with $d_{\max lc}$, but with more restrictive ones. Backtracking to other restrictions is used if no solution is found. For localization tasks, the situations is different: One wants to have an overview about all possible poses.

Furthermore, if a classical constraint problem is inconsistent, then the problem has no solution. In localization problems, there does exist a solution in reality – the real poses of the objects under consideration. The inconsistency is caused, e.g., by noisy sensory data. Therefore, some constraints must be relaxed or enlarged in the case of inconsistencies. This can be done during the propagation process by the choice of even larger restrictions than given by the maximal locally consistent restriction function.

Definition 6.4.3 *(Conservative propagation function) A propagation function $d : 2^U \times \mathcal{C} \rightarrow 2^U$ is called* **c***onservative if $D \cap C \subseteq d(D, C)$ for all D and C.*

Note that the maximal locally consistent restriction function $d_{\max lc}$ is conservative. We have:

Proposition 6.4.1 *Let the propagation function d be conservative.*

1. *Then it holds for all restrictions $D_i : \bigcap \mathcal{C} \subseteq D_i$.*

2. *If any restriction D_i is empty, then there exists no solution, i.e. $\bigcap \mathcal{C} = \emptyset$.*

Conservative functions have been used earlier for different kinds of sets, e.g., the Chebychev circular center algorithms, finding the smallest hypersphere, that contains a result set generated from constraint intersections [30], but their calculation can be complex. If no solution can be found, then the constraint set is inconsistent. There exist different strategies to deal with that as was published in [43]:

- enlargement of some constraints from $\mathcal{C}$,

- usage of only some constraints from $\mathcal{C}$,
- computation of the best fitting hypothesis according to $\mathcal{C}$.

As mentioned above, intervals are often used for the restrictions D, since the computations are much easier. Constraints are intersected with intervals, and the smallest bounding interval can be used as a conservative result. Examples are given in Fig. 6.4.

Definition 6.4.4 *(Interval Propagation)*

1. *A propagation function d is called an **interval propagation function** if the values of d are always intervals.*
2. *The **minimal conservative interval propagation function** $d_{\min c} : 2^U \times \mathcal{C} \to \mathcal{I}$ is defined by $d_{\min c}(D, C) := \min\{I | I \in \mathcal{I} \wedge D \cap C \subseteq I)\}$ for all D and C.*

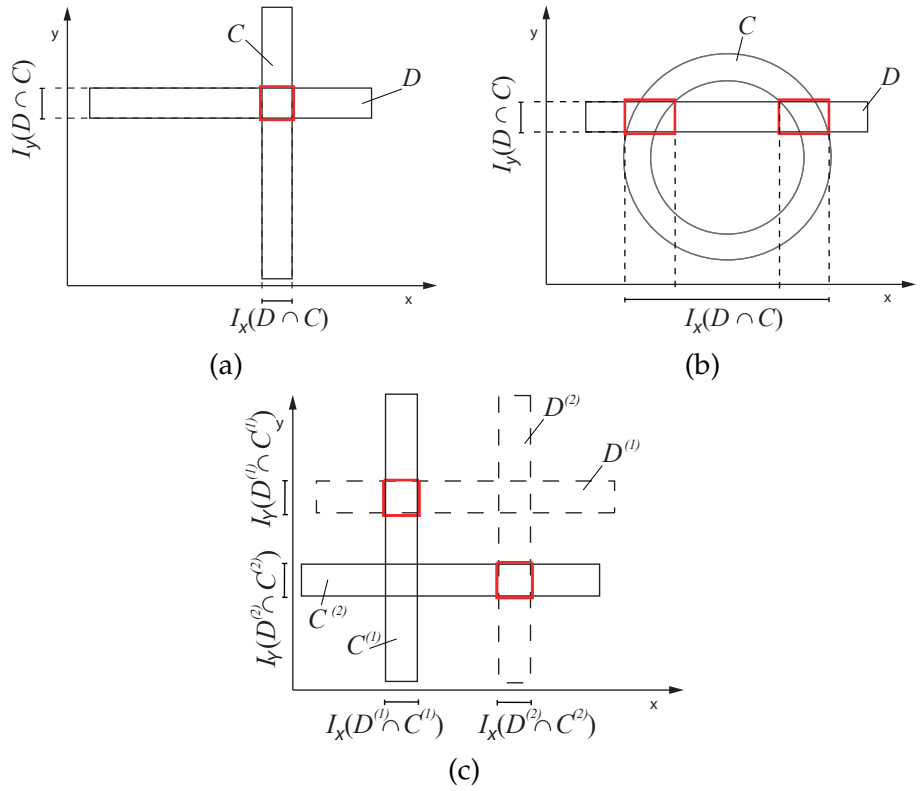


Figure 6.4: Constraint propagation with conservative intervals D. Propagations of: (a) A rectangular constraint C, (b) a circular constraint C. *Intervals of Projection* w.r.t. $C \cap D$ with another rectangular constraint are illustrated. (c) Two constraints consist of two box constraints. They are propagated with each other, which results in a new constraint (depicted by the small bold rectangles).

The results by minimal conservative interval propagation functions can be computed using projections.

Definition 6.4.5 *(Interval of projection)*

The (one-dimensional) ***I****nterval of projection w.r.t. to a set $M \subseteq U$ for a variable v is defined as the smallest interval containing the projection $\Pi_v(M)$ of M to the variable v: $I_v(M) = \min\{I | I \subseteq \mathbb{R} \wedge \Pi_v(M) \subseteq I\}$. It can be computed as $I = [a, b]$ with $a := \min(\Pi_v(M))$ and $b := \max(\Pi_v(M))$.*

Both, maximal local consistency and minimal conservatism lead to the same results, and both can be computed using the projections (Fig. 6.4):

Proposition 6.4.2

1. $d_{\max lc}(D, C) = d_{\min c}(D, C)$

2. $d_{\min c}(D, C) = I_{v(1)}(D \cap C) \times \cdots \times I_{v(k)}(D \cap C)$.

While local consistency is the traditional approach (to find only some solutions), the approach with conservative intervals is more suited for localization tasks because it can be modified w.r.t. to enlarging constraints during propagation for preventing from inconsistency. In case of inconsistencies, the algorithm below would be modified accordingly in step 6.

The following simple and practicable algorithm is used for propagation, see Alg. 4. The stop condition compares the progress after processing each constraint once. Since stabilization needs not to occur, an additional time limit is provided. Note that the step counting s is not identical to step i in the basic scheme BSCP, but could be arranged accordingly.

Constraint Hierarchies

In some cases, as in Fig. 6.4 the intersections between two constraints can be better approximated by a set of hyper-intervals instead of using just one hyper-interval.

Therefore, the *constraint union*, also *box union constraint* was introduced, which means that a constraint consists of a set of hyper-intervals, and all hyper-intervals are of the same dimension, as in Fig. 6.4 (b) and (c). By doing this, the constraint union contains fewer points that do not belong to the original constraints than when using a single box

constraint.

Algorithm 4: Constraint Propagation with Minimal Conservative Intervals, MCI-algorithm

Input: constraint set $\mathcal{C} = \{C_1, \ldots, C_n\}$ with variables $\mathcal{V} = \{v_1, \ldots, v_k\}$ over domain U and a time bound T
Data: $D \leftarrow U$, $s \leftarrow 1$, $D_{old} \leftarrow \emptyset$
Result: minimal conservative k-dimensional interval D

```
1  while s < T & D ≠ D_old do
2      foreach C ∈ C do
3          foreach v ∈ V do
4              D(v) ← I_v(D ∩ C);
5          end
6          D ← D(v_1) × ··· × D(v_n);
7      end
8      D_old ← D;
9      s ← s + 1;
10 end
```

6.5 Quality Measures for Constraint Sets

If all sensory measurements are precise, it is probable to get a well defined solution for the navigation problems, i.e., exact positions of the interesting objects. Furthermore, there must be enough constraints to restrict the possible solutions to a small constraint set. Otherwise one has to deal with ambiguities. Hence there are two aspects for the quality of a constraint set: Its consistency and its ambiguity. Both are subject to trade offs (cf. Section 6.6). Thus, measures for inconsistency and ambiguity are defined now. Thereby the main focus is put on definition of those measures, the discussion of possibilities of their efficient calculation is left open right now.

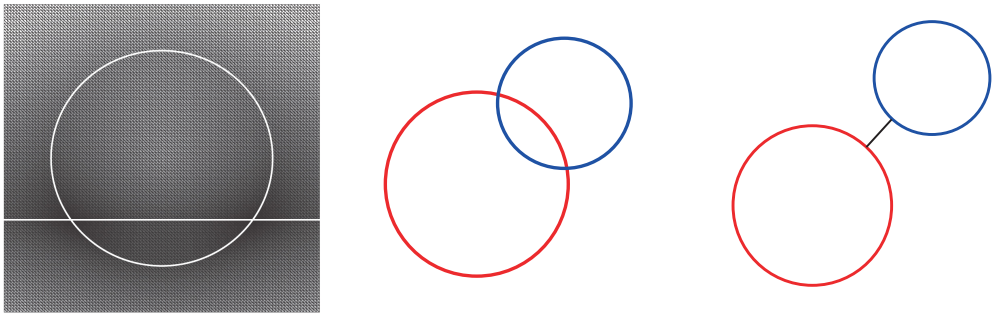
6.5.1 Inconsistency Measure

A constraint system $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$ is inconsistent if there is no solution according to Definition 6.1.2, i.e., if $\bigcap \mathcal{C}$ is empty. Now a measure for inconsistency should reflect "how far" $\mathcal{C}$ is from having a solution.

Definition 6.5.1 *(Distances)*
Let $d(p, p')$ be a distance between points p and p' in the universe U (we use the Euclidean distance in our examples).

1. *The distance between any point $p \in U$ and a constraint $C \subseteq U$ is given by*

$$d(p, C) = \min_{p_c \in C} \{d(p, p_c)\}, \tag{6.6}$$



(a) (b) (c)

Figure 6.5: Inconsistency measure example: (a) The points of the same gray value have the same sum of distances s to the two constraints, constraints are depicted as white circle and white line. The smaller s is, the darker are the points. The set $P_{\mathcal{C}}(s)$ consists of all points where the sum of distances is smaller or equal than s. (b) and (c) give examples for IC. Left: $IC(\mathcal{C}) = 0$. right: $IC(\mathcal{C}) > 0$.

2. *The distance between any point $p \in U$ and a set of constraints $\mathcal{C} = \{C_1, \ldots, C_n\}$ is given by*

$$d(p, \mathcal{C}) = \sum_{i=1}^{n} d(p, C_i) \tag{6.7}$$

3. *The set of all points $p \in U$ where the distance to the constraint set $\mathcal{C}$ is equal or less s is defined by*

$$P_{\mathcal{C}}(s) = \{p | d(p, \mathcal{C}) \leq s\} \tag{6.8}$$

There are visualizations of sets $P_{\mathcal{C}}(s)$ on Fig. 6.5. As a corollary we find:

Corollary 6.5.1 *A constraint set $\mathcal{C}$ is consistent iff $P_{\mathcal{C}}(0)$ is not empty.*

This leads to the following definition of the inconsistency measure for constraint sets $\mathcal{C}$:

Definition 6.5.2 *(Inconsistency Measure)*

$$IC(\mathcal{C}) = \min(\{s | P_{\mathcal{C}}(s) \neq \emptyset\}) \tag{6.9}$$

The geometrical interpretation of the inconsistency measure is the shortest distance the constraints of the system $\mathcal{C}$ have to be moved to in order this system to become consistent:

Corollary 6.5.2

$$IC(\mathcal{C}) = \min_{p} \sum_{i=1}^{n} d(p, C_i) \tag{6.10}$$

Examples with two circular constraints are given in Fig. 6.5. If both circles have intersecting points (b), then the system is consistent with $IC = 0$. But there might be some ambiguity as in Fig. 6.5 (b). These situations are discussed in the following subsection. Fig. 6.5 (c) depicts an inconsistent system with $IC > 0$. All points on the black line between points a and b have the same sum of distances to the constraint set. We have $IC(\mathcal{C}) = d(a, b)$ and $P_{\mathcal{C}}(IC(\mathcal{C}))$ is the line between a and b.

In case of global inconsistency, i.e., $IC(C) > 0$, the constraint set $\mathcal{C}$ has no solution, but for small values of $IC(\mathcal{C})$ we could use $P_{\mathcal{C}}(IC(\mathcal{C}))$ instead. For larger inconsistencies, one could try to find some consistent subsets of the constraint set. But it can happen that the solution becomes more ambiguous. In the following a first ambiguity measure for constraint sets is proposed.

6.5.2 Ambiguity Measures

Without constraints, any $p \in U$ is a possible solution, and having only a single constraint C , all $p \in C$ are candidates for the solution of the navigation problem. Differently to other constraint satisfaction problems, the robot has a certain position in reality – this is what should be estimated. If the constraints do not allow an exact determination of the position, we are left with some ambiguity. Actually, the robot needs to solve its navigation problem in order to fulfil some tasks. Therefore, some ambiguity may be without consequences. It might be enough to know that the robot is in a certain area (e.g. for avoiding offside in soccer).

On the other hand, having an inconsistent constraint set $\mathcal{C}$, there is no candidate for a solution. But since we know that the robot is somewhere in the environment, one can look for a nonempty set $P_{\mathcal{C}}(s)$. Those sets with minimal parameter s would be the best guesses for the navigation problem, especially $P_{\mathcal{C}}(IC(\mathcal{C}))$.

Hence a definition of an ambiguity measure for any subset $P \subseteq U$ is given. There are different aspects to be considered. The volume of P could be a measure for the total amount of possible solutions. If P is disconnected, then the number of connected components, or the distance between the components provide other measures. As discussed above, it depends on the task of the robot and on the situation which measure is adequate. Actually, one property which is important for our results is monotonicity: The ambiguity increases if the set P becomes larger.

Definition 6.5.3 *(Ambiguity)*
For a nonempty set $P \subseteq U$ the ambiguity is defined by:

$$Amb(P) = \max\{d(p, p') | p, p' \in P\} \tag{6.11}$$

For technical reasons, we define $Amb(\emptyset) \triangleq -1$ for the empty set.

By this definition, the ambiguity is zero if and only if P contains a single point.

6.6 Optimal Constraint Sets

Using more constraints can reduce ambiguity but increases inconsistency: There is a trade-off which affects the use of available constraints. In fact, a lot of different sensory data and other information can be used for navigation purposes, but they may contradict each other to some extent by sensory noise or processing errors. It may be useful to accept a certain degree of inconsistency for having less ambiguity. As discussed in Section 6.5.2, it depends on the tasks of the robots which kind of ambiguity is acceptable.

Therefore, it is necessary to evaluate constraint sets $\mathcal{C}$ in order to decide for using more or less constraints. The consequences of changing constraint sets $\mathcal{C}$ for inconsistency and ambiguity are considered now. Obviously, the set of solutions decreases for more constraints:

$$\mathcal{C} \subseteq \mathcal{C}' \Rightarrow \bigcap \mathcal{C} \supseteq \bigcap \mathcal{C}' \tag{6.12}$$

More generally, we have for any number s:

$$\mathcal{C} \subseteq \mathcal{C}' \Rightarrow P_{\mathcal{C}}(s) \supseteq P_{\mathcal{C}'}(s) \tag{6.13}$$

Concerning our quality measures we have:

Proposition 6.6.1

$$\mathcal{C} \subseteq \mathcal{C}' \Rightarrow IC(\mathcal{C}) \leq IC(\mathcal{C}') \tag{6.14}$$

$$\mathcal{C} \subseteq \mathcal{C}' \Rightarrow Amb(\bigcap \mathcal{C}) \geq Amb(\bigcap \mathcal{C}') \tag{6.15}$$

The proposition marks a classical trade-off between inconsistency and ambiguity for the choice of constraints to be used: The larger one chooses the constraint set $\mathcal{C}$ from a set of available constraints, the more decreases the ambiguity of the solution set and the more increases the inconsistency of the system. Actually the solution set becomes empty (and ambiguity becomes simply -1) if inconsistency reaches a value greater than zero (if the system becomes inconsistent). Hence, considering only this trade-off is not really helpful for our purposes.

For our purposes, the sets $P_{\mathcal{C}}(IC(\mathcal{C}))$ are more important since one can use them as substitutes for the solution sets $\bigcap \mathcal{C}$ in case of inconsistent constraint sets $\mathcal{C}$. At first sight, it might seem that the sets $P_{\mathcal{C}}(IC(\mathcal{C}))$ decrease as well if the constraint sets $\mathcal{C}$ become larger in a similar way as for the solutions in (6.12) from above. Then the ambiguity of these sets would decrease as well similarly to (6.15). But the sets $P_{\mathcal{C}}(IC(\mathcal{C}))$ do not necessarily decrease if the constraint sets become larger:

Proposition 6.6.2 *There exist constraint sets $\mathcal{C} \subseteq \mathcal{C}'$ such that $P_{\mathcal{C}}(IC(\mathcal{C})) \subseteq P_{\mathcal{C}}(IC(\mathcal{C}'))$ and hence $Amb(P_{\mathcal{C}}(IC(\mathcal{C}))) \leq Amb(P_{\mathcal{C}}(IC(\mathcal{C}')))$.*

As an example, consider constraint sets $\mathcal{C} = \{\{p\}\}$ and $\mathcal{C}' = \{\{p\}, \{p'\}\}$ for different points $p, p' \in U$. Then we have $IC(\mathcal{C}) = 0$ and $IC(\mathcal{C}') = d(p, p')$, respectively. Thus, we get: $P_{\mathcal{C}}(IC(\mathcal{C})) = \{p\} \subseteq \{p, p'\} \subseteq P_{\mathcal{C}'}(IC(\mathcal{C}'))$ which shows the proposition.

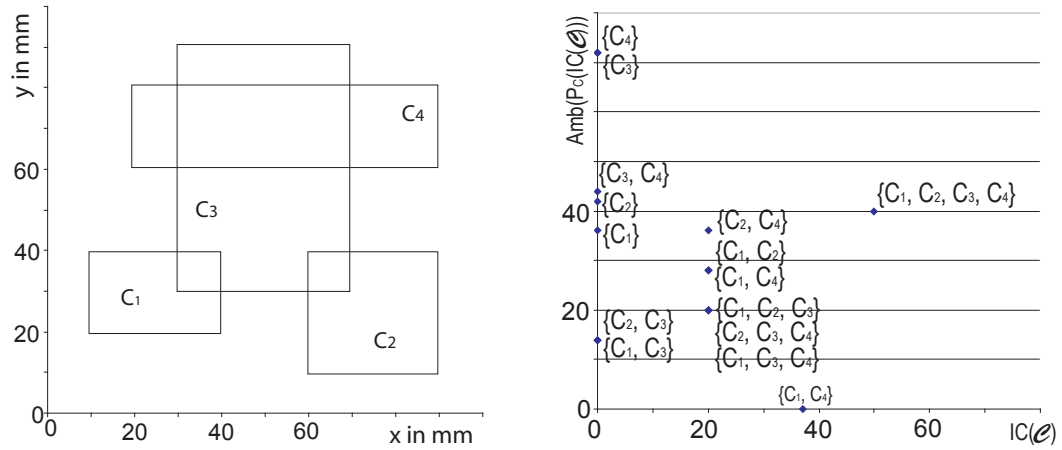
To go more into the details: We have indeed $P_{\mathcal{C}}(s) \supseteq P_{\mathcal{C}'}(s)$ for $\mathcal{C} \subseteq \mathcal{C}'$ by (6.13), but now the values of s may increase if we use $s = IC(\mathcal{C})$ for inconsistent constraint sets. For such cases we have:

Proposition 6.6.3

$$s \leq s' \Rightarrow P_{\mathcal{C}}(s) \subseteq P_{\mathcal{C}}(s') \tag{6.16}$$

$$s \leq s' \Rightarrow Amb(P_{\mathcal{C}}(s)) \leq Amb(P_{\mathcal{C}}(s')) \tag{6.17}$$

Fig. 6.6 presents on the right the values of $IC(\mathcal{C})$ and $Amb(P_{\mathcal{C}}(IC(\mathcal{C})))$ for all nonempty subsets $\mathcal{C}$ of the constraint set $\mathcal{C}' = \{C_1, \ldots, C_4\}$ from the left. The values for the sequence $\{C_1\} \subset \{C_1, C_2\} \subset \{C_1, C_2, C_3\} \subset \{C_1, C_2, C_3, C_4\}$ do not lie on a monotonous line in the diagram: Using more constraints may increase the value of the ambiguity measure $Amb(P_{\mathcal{C}}(IC(\mathcal{C})))$. The example illustrates the fact that a more detailed analysis is necessary to find an optimal set of constraints.



(a) (b)

Figure 6.6: Inconsistency and ambiguity correlation: (a) Example constraint set $\mathcal{C}$. Each constraint consists of the boarder points of a rectangle. (b) Inconsistency and ambiguity measure for elements of the potential set of $\mathcal{C}$.

At the end of this section shall be remarked that the results for ambiguity use only the monotony property: If $P \subseteq P'$, then $Amb(P) \leq Amb(P')$. Therefore, the results would be true for other kinds of ambiguity measures, too.

6.7 Handling Inconsistencies

Noisy robot data, especially from real robots can lead to inconsistent constraints, i.e., no global solution can be found. But there exists a variety of possibilities to attack this problem. In this section the following strategies are discussed: *greedy propagation,*

sensory data clustering, sensor resetting, constraint border enlargement and *soft-cuts*. Some of these approaches can be combined. Greedy propagation provides a simple heuristic for the constraint propagation order. Soft-cuts constitute an approach to generate a constraint from two others as an alternative to the constraint intersection. Sensor resetting can be used when the number of inconsistent constraints exceeds a certain threshold. Sensory data clustering focusses on finding consistent constraint subsets.

Algorithm 5: Greedy propagation

Input: $C_t^{B-}, C_t^{z_1}, \ldots, C_t^{z_n}$
Result: C_t^B

```
1 C_t^B ← C_t^{B-} ;
2 for i = 1 to n do
3 |   S ← C_t^B ∩ C_t^{z_i};
4 |   if S ≠ ∅ then
5 |   |   C_t^B ← S ;
6 |   end
7 end
8 return C_t^B
```

6.7.1 Greedy Propagation

At first one has to consider, which constraints shall be propagated with each other. The constraint that represents the robot's current belief at time t is denoted by C_t^B. Furthermore, the constraints that are generated from sensory data are denoted by $C_t^{z_1}, \ldots, C_t^{z_n}$, where $z_1, \ldots, z_n$ stand for the different sensory data. The propagation order of the constraints has to be decided and if inconsistencies occur the algorithm has to determine which constraints to relax. A greedy approach is to iteratively propagate the current belief with the sensory data while the resulting constraint is not empty, as in Alg. 5. The advantage of this approach is its simplicity. One disadvantage is that whenever two sensory data constraints are inconsistent with each other, the later propagated constraint will be discarded. Thus, the order of constraint propagation affects the resulting constraint C_t^B. Consequently there can be different consistent subsets for different propagation orders.

Another approach is to find a maximal subset of the sensory data constraints, as described in Section 6.7.2. The sensory data constraints $C_t^{z_i}$ are propagated with each other at first and then the resulting constraint S is propagated with the a-priori belief constraint C_t^{B-}, see Alg. 6.

If S and the a-priori belief constraint C_t^{B-} are not empty, i.e., $C_t^{B-} \cap S \neq \emptyset$, they are propagated, resulting in the a-posteriori belief constraint C_t^B. If the sensory data constraint result S is empty or has no common elements with C_t^{B-}, i.e., $C_t^{B-} \cap S = \emptyset$, the boundaries of C_t^{B-} are increased and C_t^B takes the value of the increased C_t^{B-}. Experimental data showed an improved convergence using this approach. The disadvantage

is that the algorithm does not analyze, which sensory data constraints fit better and which fit worse to the rest of the constraints.

An approach to find consistent sets of sensor readings is to cluster sensory data.

Algorithm 6: Sensory Data Constraints Propagation

Input: $C_t^{B-}, C_t^{z_1}, \ldots, C_t^{z_n}$
Result: C_t^B

```
1  S ← C_t^{z_1} ;
2  for i = 2 to n do
3  |  S ← S ∩ C_t^{z_i};
4  end
5  if S ∩ C_t^{B-} ≠ ∅ then
6  |  C_t^B ← C_t^{B-} ∩ S ;
7  else
8  |  C_t^B ← increase_Boundaries(C_t^{B-})
9  end
10 return C_t^B
```

6.7.2 Sensory Data Clustering

A method to find sensory data outliers is to create sensory data subsets $\mathcal{C}_{s_i} \subseteq \mathcal{C}$ and to check, if the resulting constraint is not empty. Theoretically one has to calculate for every element of the potential set of $\mathcal{C}$ if it is consistent. But the number of elements in the potential set of $\mathcal{C}$ grows exponentially with the number of elements in $\mathcal{C}$. A faster and more efficient strategy is to calculate all subsets $\mathcal{C}_{s_k} \subseteq \mathcal{C}$ where $\text{card}(\mathcal{C}_{s_k}) = \text{card}(\mathcal{C}) - 1$. This means that only those possible subsets of $\mathcal{C}$ are generated, which have one element less, i.e. one constraint less than $\mathcal{C}$. There are $\text{card}(\mathcal{C})$ of those subsets that are missing only one constraint, in general there exist:

$$s = \begin{pmatrix} \text{card}(\mathcal{C}) \\ m \end{pmatrix} = \frac{\text{card}(\mathcal{C})!}{m! \cdot (\text{card}(\mathcal{C}) - m)!}$$

possible subsets of $\mathcal{C}$ with m elements.

The calculation of the resulting restriction for a given constraint subset can be computationally expensive. Assuming n is the number of constraints in $\mathcal{C}$, then one has to perform $n - 1$ propagation steps for every subset. With n possible subsets we have to perform $n \cdot (n - 1)$ propagations. Thus, the asymptotic complexity is $O(n^2)$.

To reduce this complexity in cases with many constraints, binary trees are used, where the leaves contain all the sensory data constraints and other nodes contain constraints generated by propagation of their two child constraints. Fig. 6.7 (a) gives an example for a tree, i.e., two subtrees generated from eight sensory data constraints. For a given set of constraints, one can efficiently use the constraint tree. The tree has $O(n)$ elements which can be calculated by $O(n)$ propagation steps. For a given constraint

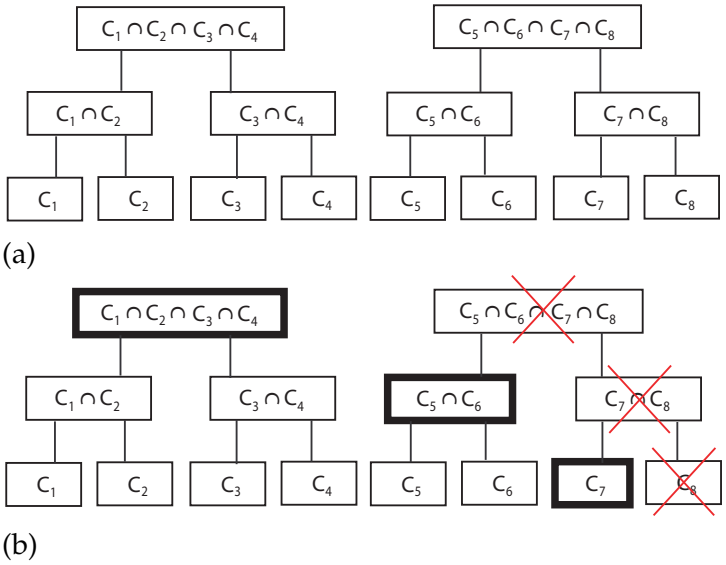


Figure 6.7: Constraint clustering tree: (a) Containing sensory data constraints as leaves and other nodes as results from two propagated constraints; (b) example to generate the resulting restriction from $\bigcap(\mathcal{C} - C_8) = C_1 \cap C_2 \cap C_3 \cap C_4, C_5 \cap C_6, C_7$.

subset, one has to propagate $\log(n)$ constraints, one constraint in each tree level, cf. Fig. 6.7 (b). The complexity for the whole algorithm is reduced to $O(n \log(n))$.

6.7.3 Model Resetting

If there are outliers, i.e., there are inconsistent sensory data with respect to the belief constraint, different possibilities exist to treat them. An easy way is to reject the outliers. But in case of a kidnapped robot, the robot then has no possibility to determine its new position, since all new sensory data are rejected only with respect to the old belief. A better possibility is to remember the outliers and if too many of them occur to create a new model, i.e., a new constraint belief from the outlier constraints. If two belief constraints are similar, they can be merged into one constraint.

This approach is somewhat similar to the model splitting and merging of Gaussians in MHT, as described in [3]. A heuristic to distinguish noisy sensory data from sensory data that results from a kidnapped robot is to count the number of inconsistent sensor readings n_{inc} for a certain time t_{inc}, e.g., 10 seconds. If no inconsistent sensor reading occurred during this time, n_{inc} becomes zero. The number of consistent sensor readings n_{con} is counted as well. Now we can calculating the *inconsistent sensory data rate* (IDR), where $IDR = \frac{n_{inc}}{n_{inc}+n_{con}}$. If this rate becomes larger than a threshold, a new belief constraint is generated from the last perceived sensory data. Each belief constraint is maintaining its own IDR. The inconsistent data rates are used to decide, which belief constraint is most consistent (has the smallest IDR) and will be used for calculating

the current robot position. To avoid oscillations, a hysteresis function is used to switch between two constraints. Assume C_1 is the current belief constraint with IDR_1, one switches to C_2 as soon as $IDR_2 + \varepsilon < IDR_1$. Belief constraints, that have not been used over a long time are discarded. This approach of neglecting sensory data which does not match the current belief is a bit similar to the approach of Burgard [8], who used an entropy filter and two buckets to decide which sensory data to take and which to reject.

6.7.4 Constraint border enlargement.

This subsection describes another algorithm to handle inconsistent sensory data. Within every prediction step, the constraint borders of the belief constraint are enlarged to account for walking errors. If consistent sensory data is perceived, it leads to a reduction of the belief constraint. If the sensory data are inconsistent the constraint is not affected by the intersection and its borders are increased for the next time frame. If sensory data are inconsistent over a long time period, e.g., because the robot was kidnapped, the constraint boarders are enlarged to the whole state space, i.e., the whole field size. The belief constraint is then propagated with the current sensory data constraints, cf. Alg. 6.

6.7.5 Weighted Soft-Cuts and Intra-Constraint Merges

Soft-Cuts. A further method to propagate constraints with each other, is called *Soft-Cut*. Its main advantage is its usability in situations with high sensory noise. In those situations, constraint intersections can be empty. In contrast to usual constraint intersections, soft-cuts do not calculate the conservative intersection of two constraints. Moreover, they calculate the bounding vectors as a weighted average of two constraint elements (e.g., boxes). One has to remember that constraints can be described as a union of boxes (or hyper-intervals). Each n-dimensional box can be described by a $2n$-dimensional vector that contains two numbers per dimension. These two numbers encode the interval boundaries in the corresponding dimension. Soft-cuts then calculate the weighted average between two constraints, by calculating the weighted average between the two vectors, as presented in Fig. 6.8 (a). Soft-cutting two box union constraints can be decomposed to soft-cutting each box T_i of the first constraint with each box T_j of the second constraint, cf. Fig. 6.8 (c). Let C_1 and C_2 be two box union constraints of dimensionality n with $T_i \in C_1$ with $i \in \{1, \ldots, k\}$ and $T_j \in C_2$ with $j \in \{1, \ldots, l\}$. Then $C_1 \cap_{soft} C_2$ is defined as:

$$C_1 \cap_{soft} C_2 = \{C | C = \bigcup_{T_i \in C_1, T_j \in C_2} T_i \cap_{soft} T_j\} \tag{6.18}$$

With $T_i \cap_{soft} T_i$ is defined as:

$$T_i \cap_{soft} T_j = \{(s_1, \ldots, s_n, f_1, \ldots, s_n) = T_R | T_R = \alpha T_i + (1 - \alpha) T_j | \alpha \in [1, 0]\} \tag{6.19}$$

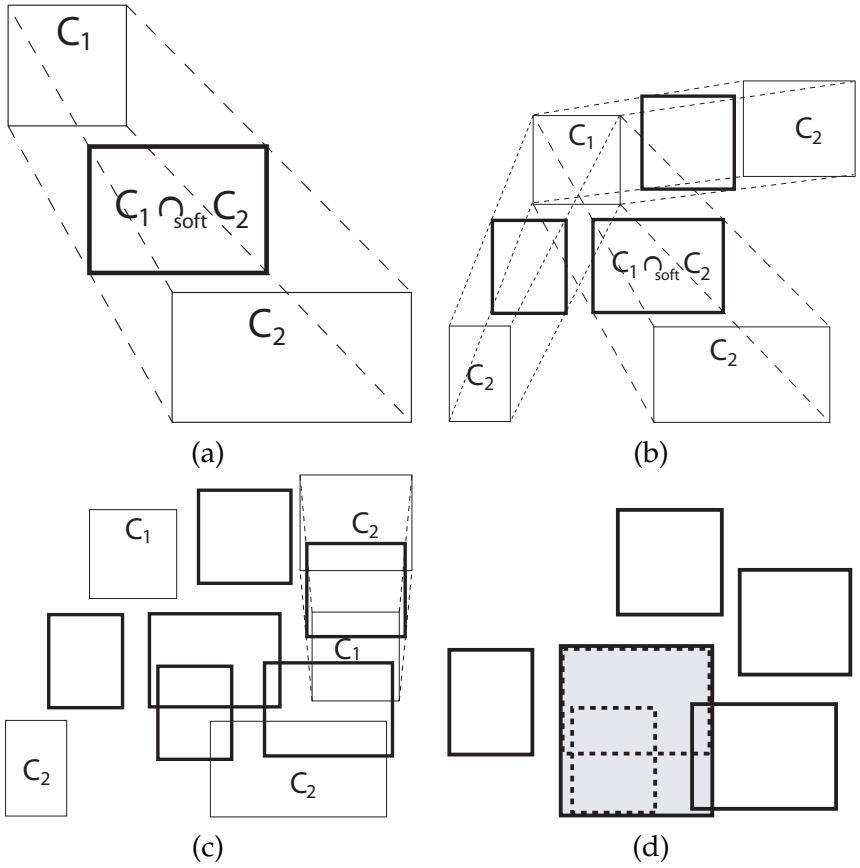


Figure 6.8: Constraint soft-cuts and merges. (a) Two box constraints are soft-cut with each other. (b) A single box constraint is soft-cut with a box union constraint. (c) Two box union constraints are soft-cut, resulting in a box union with six elements. (d) Merging example, two strongly overlapping boxes are merged to a new box.

Depending on the number of boxes within each constraint union, the resulting constraint from a soft-cut can have a high number of elements, thus the practical applicability of soft-cuts has to be tested in the next chapter.

Intra-Constraint Merges. One disadvantage of soft-cuts is that the cardinality of the resulting box union constraints is growing quickly. A constraint, that was calculated by soft-cutting two constraints C_i and C_j has the following number of elements:

$$\text{card}(C_i \cap_{soft} C_j) = \text{card}(C_i) \cdot \text{card}(C_j) \tag{6.20}$$

In analogy to Multi-Hypotheses Tracking, one has to find a solution to decrease the number of elements within a constraint union. This can be done by *intra-constraint merges*. The idea is to merge boxes within a constraint, that are overlapping and replace

those boxes by a single constraint. Therefore, a measure is necessary to decide which constraints should be merged. One possibility is to consider the volume of two constraint candidates before and after the merge. If the volume of the merged constraint is not bigger than the sum of the original constraints multiplied by a certain threshold factor, the merge is applied. The merge of two constraints can be, e.g., the conservative interval, containing both original constraints.

6.8 Position Estimate

Estimating the position is a non-trivial problem for constraints, especially in case of a multi-modal belief distribution. In some cases constraints consist of several constraint parts. The philosophy behind this is that in some cases the sensory data does not allow restricting the set of possible robot positions. Then it is useful to say that the robot does not know about its position, e.g., when the robot is perceiving a line only. Consequently the next layer, usually the behavior control has to decide how to handle the modeling uncertainty. But often the robot can restrict the set of possible positions, so that a minor ambiguity remains. Then we want to estimate the most probable position, given the belief constraint. One method is to use the weighted sum of the centers of gravity of all constraint parts. The disadvantage is that the resulting robot position estimate does not necessarily have to lie within the constraint by which it was generated. To avoid this problem, one could try to find a point within the constraint. When having a union of box constraints one could take the center point of the biggest box, which was successfully done by us in [45]. In many cases it is useful to combine the modeled position with a confidence measure to tell the next layer, e.g., the planning layer how accurate this estimation is.

Confidence Measures. In Section 3.6 the entropy was presented as a convergence measure for particle filters. When using Kalman filters, the covariance matrix serves as a measure for the uncertainty of the modeled position. For constraints it is possible to calculate the entropy as well, using a grid. Another method is, e.g., to use an ambiguity measure as described earlier or to use the volume of the constraints, which in some cases can be calculated very efficiently. High constraint volumes indicate that the position ambiguity is very high. This constraint volume measure was used in an implementation within the next chapter.

6.9 Summary

Constraint based localization techniques consist of a variety of sub-techniques to estimate the state of an object. One question is, which sensory data should be used and how to store it within the robot's memory. The kind of representation is also relevant for the belief. If possible, one can use a heuristic to discard sensory data, that is not matching the belief or the rest of sensory data. Finally one has to chose a propagation

method, which shall be efficient in computational needs and robust to sensory noise. This usually results in a tradeoff.

Not all of the introduced concepts are implemented already on a robot platform. Some measures, e.g., the measure of inconsistency IC of two or more constraints are hard to calculate with respect to calculational resources but they are of theoretical interest.

After having introduced the theory behind constraint based modeling techniques, the next chapters demonstrate how localization and object tracking approaches can be implemented on a four-legged and a two-legged robot within the RoboCup domain.

7 Constraint Based Localization

In this chapter an implementation of a constraint based localization application is introduced. In different domains, landmarks are more or less sparsely arranged. In RoboCup the number of unique landmarks has been reduced over the last years. Other sensory data as information about field lines has to be considered more for self-localization. Compared to flags, perception of a field line results in a complex belief function which is hard to represent by a Gaussian or by a small set of particles. Therefore, a constraint based self-localization implementation will be described and compared to a Monte-Carlo Localization algorithm. Some problems arising in the implementation part are specific and might be avoided with a different implementation, so the author tried to be not too restrictive with his assumptions about an implementation.

At first we will take a closer look on how constraints can be generated from sensory data, i.e., percepts within the specific RoboCup domain. Then we will analyze how different sensory data affect the geometric shape of constraints. After presenting different sensory data constraints, constraint propagation approaches are discussed and several experiments are performed. The last section summarizes the results.

7.1 Constraint Generation from Percepts

An example camera image from RoboCup is given, the image, that was perceived by a robot shows a goal, a ball, and a white line of the penalty area (see Fig. 7.1). It is not too difficult for a human interpreter to give an estimate for the position (x_B, y_B) of the ball and the position (x_R, y_R) of the observing robot. Humans can do that, regarding relations between objects, like the estimated distance d_{BR} between the robot and the ball, and by their knowledge about the world, like the positions of the goalposts and of the penalty line. The program of the robot can acquire the related features using image processing. The distance d_{BR} can be calculated from the size of the ball in the image, or from the vertical bearing angle to the ball center. The distance d_{BL} between the ball and the penalty line can be calculated, too. Other values are known parameters of the environment: $(x_{Gl}, y_{Gl}), (x_{Gr}, y_{Gr})$ are the coordinates of the goalposts, and the penalty line is given as the set of points $\{(x, b_{PL}) | -a_{PL} \leq x \leq a_{PL}\}$. The coordinate system has its origins at the center point, the y-axis points to the observed goal. The relations between objects can be described by constraints. The constraints of Fig. 7.1 can be generated automatically from the given percepts of the image:

C_1**:** The view angle γ between the goalposts (the distance between them in the image) defines a circle (periphery circle), which contains the goal posts coordinates

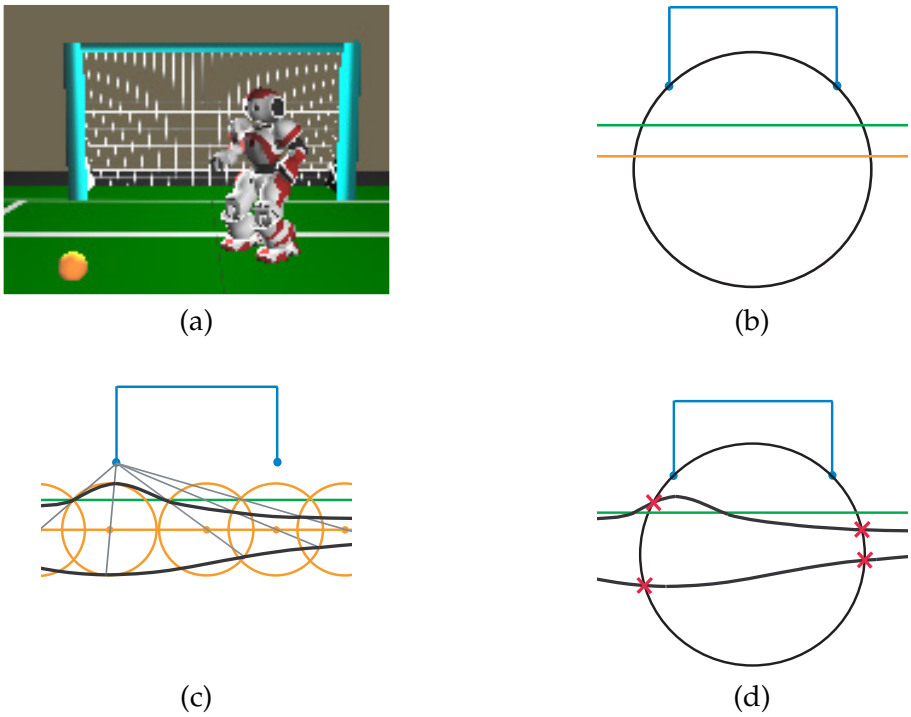


Figure 7.1: Constraint example of a game situation (Standard Platform League): A robot perceives a goal, a ball, and another robot in front of the penalty line. (a) The scene from the robot's perspective, (b) shows the resulting robot positions represented by the periphery circle according to C_1, and the shape of the ball line constraint C_2. (c) The constraint C_2 for the ball, some of the circles defined by C_5, some lines according to C_4, and the resulting shape (black) for C_6 are presented. (d) Constraints according to C_7: The position of the robot is one of the four intersections of C_1 and C_6.

(x_{Gl}, y_{Gl}), (x_{Gr}, y_{Gr}) and the coordinates (x_R, y_R) of the robot:

$$\{(x_R, y_R) | \arctan \frac{y_{Gl} - y_R}{x_{Gl} - x_R} - \arctan \frac{y_{Gr} - y_R}{x_{Gr} - x_R} = \gamma\}$$

C_2: The ball lies in the distance d_{BL} before the penalty line. Thus, the ball position must be from the set:

$$\{(x_B, y_B) | x_B \in [-a_{PL}, a_{PL}], y_B = b_{PL} - d_{BL}\}$$

C_3: The distance d_{BR} between the robot and the ball defines a circle, so that the robot is on that circle around the ball:

$$\{(x_R, y_R, x_B, y_B) | (x_B - x_R)^2 + (y_B - y_R)^2 = d_{BR}^2\}$$

C_4**:** The observer, the ball and the left goal post are on a line:

$$\{(x_R, y_R, x_B, y_B) | \frac{x_R - x_B}{y_R - y_B} = \frac{x_B - x_{Gl}}{y_B - y_{Gl}}\}$$

The points satisfying the constraints by C_1 (for the robot) and by C_2 (for the ball) can be visualized immediately on the playground as in Fig. 7.1 (b). The constraint C_3 does not give any restriction to the position of the ball. The ball may be at any position on the playground, and then the robot has a position somewhere on the circle around the ball. Or vice versa for reasons of symmetry: The robot is on any position of the playground and the ball around him on a circle. In fact, we have four variables which are restricted by C_3 to a subset of a four dimensional space. The same applies to constraint C_4.

The solution (i.e. the positions) must satisfy all four constraints. We can consider all constraints in the four dimensional space of the variables (x_B, y_B, x_R, y_R), so that each constraint defines a subset of this space.

By combining C_2 and C_3 we get the constraint $C_5 = C_2 \cap C_3$ where the ball position is restricted to any position on the penalty line, and the player, i.e., a *soccer playing robot* is located on a circle around the ball. Then, by combining C_4 and C_5 we get the constraint $C_6 = C_4 \cap C_5$ which restricts the positions of the robot to the two lines, see Fig. 7.1 (c).

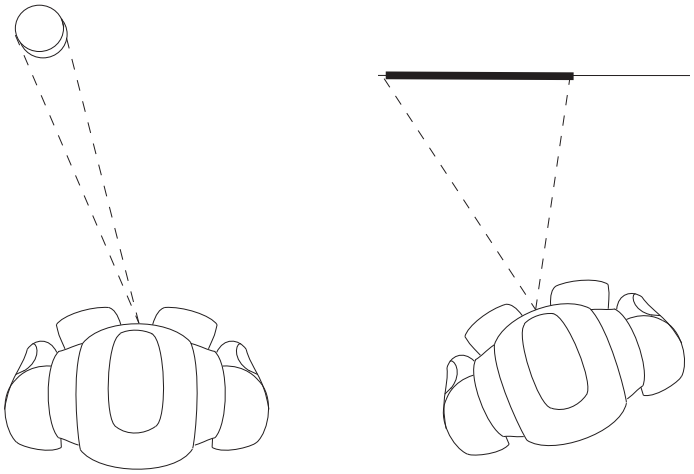
Now intersecting C_1 and C_6 we get the constraint C_7 with four intersection points, cf. Fig. 7.1 (d). According to the original constraints C_1 to C_4, these four points are determined as possible positions of the robot. The corresponding ball positions are then given by C_2 and C_4.

To find the real positions, one would need additional constraints from the image, e.g., that the ball lies between the robot and the goal (which removes one of the lines of C_6), and that the robot is located on the left site of the field (by exploiting perspective).

The example demonstrates that constraints can be applied for object modeling, self-localization, and for modeling the relations between different objects. Now shall be analyzed which sensory data can be used for object modeling and localization within the Four-Legged and the Standard Platform League (SPL).

Flag Constraints. Flags have been removed in the SPL but were used over a long time in the Four-Legged League. Whenever a robot perceives a flag or a goal post, it actually perceives the two limiting angles to that flag, see Fig. 7.2 (a), or respectively the goal posts. It can generate a constraint consisting of three variables - two for the position and one variable for the orientation. Flag constraints are ring shapes in 2-d space, the circular form is caused by the position ambiguity. For every 2-d position within the ring exists a corresponding angle interval for the robot's orientation. A similar constraint can be generated when the robot perceives the angle between two goal posts, although with other angle intervals for each position.

Line Constraints. Whenever a robot perceives a line, i.e., a line segment, it actually perceives the relative distance vector to both delimiting points of the line segment, see



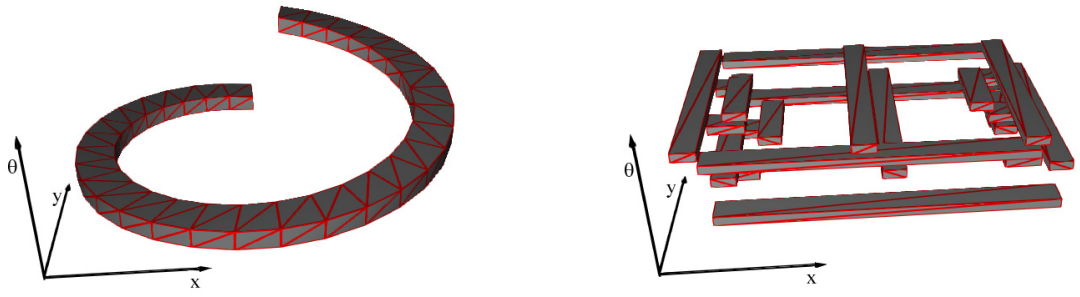
(a) (b)

Figure 7.2: Perception in RoboCup: Perception of (a) a flag and (b) a line.

Fig. 7.2 (b). The resulting constraint is defined as the union of all possible positions from where the robot can perceive this line. The structure of a line constraint C_l (index l stands for *line*) can be realized as the union of multiple three-dimensional intervals C_i, as described already in Section 6.4. We have:

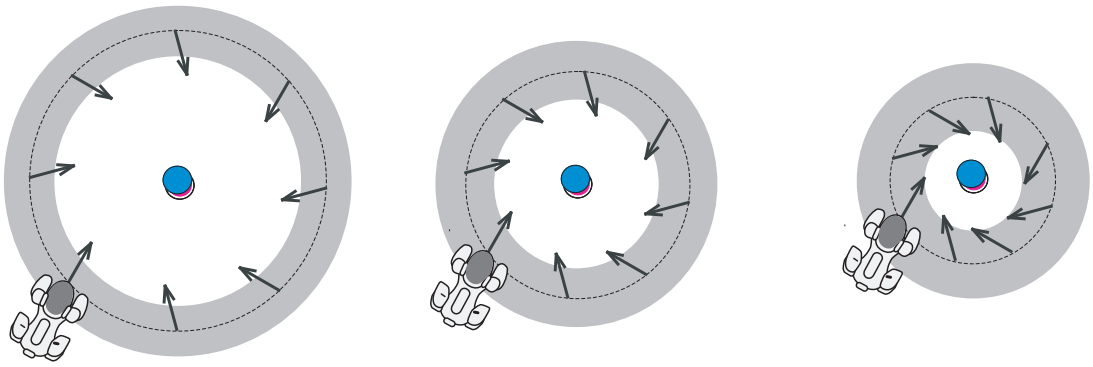
$$C_l = \bigcup_{\forall C_i \in \mathbf{C}} \mathrm{Dom}_{C_i}(x) \times \mathrm{Dom}_{C_i}(y) \times_{C_i} (\theta) \tag{7.1}$$

The representation of such a three-dimensional interval, parallel to the coordinate frame is easy. It can be represented as a vector, that contains the starting and the finishing values for all dimensions, which results in two variables for each dimension and thus, six variables in the whole vector. Fig. 7.3 presents the shape of a flag and a line constraint



(a) (b)

Figure 7.3: Constraint shapes in 3-d space, x, y, θ coordinates: (a) Circular constraint shape, (b) an example line constraint shape for a perceived line segment.



(a) (b) (c)

Figure 7.4: Updating constraints by motion data: (a) A robot is perceiving a flag, creating a constraint using the flag distance and the angle. (b) The robot is walking forward, no further perception, the constraints (distance, angle and boundaries) are propagated. (c) After another step the uncertainty increases more.

in the x, y, θ - space.

Constraint Prediction Step

The robots are moving, their positions within the environment and their distance to objects change. Also a moving object changes its position over time. In both cases the robot has to update its belief. When odometry information is available the robot can use this data to predict its belief. If it is tracking an object, the robot can use the estimated object speed, if available, to predict the new object state.

If the robot has odometry information, it can consider its motions for constraint prediction updates. As an example, when a robot perceives a flag in a certain distance and angle and the robot is moving, the distance and angle of the flag to the robot change. The robot's uncertainty about the flag distance and angle increases, due to motion noise or slippage [92]. Fig. 7.4 shows how a constraint prediction by odometry looks graphically. A modification of the constraint was implemented by updating the constraint borders with regard to the motion direction. Also the increased uncertainty has to be considered, so the constraint borders are increased by a certain δ_i for each dimension i, depending on the motion error for each dimension.

7.2 Constraint Propagation

After the introduction of different constraint types it shall be described how different constraints can be propagated with each other. More specific, when a robot is generating constraints from sensory data, it has to use those constraints to correct, i.e., to restrict its belief constraint. In this section shall be demonstrated how different constraints can

be propagated with each other. An important aspect is the efficient calculation of the resulting constraint because calculation time is an important factor on a mobile robot. The following example illustrates the difficulties, that can occur while propagating two constraints. Imagine a robot, that is perceiving two flags. The robot can generate two circular constraints and calculate the resulting set of possible positions.

For two circular constraints many different intersection shapes are possible, depending on the size and position of the circular constraints. The different combinations for intersection shapes are presented in Fig. 7.5. As one can see, the resulting intersection areas are non-trivial and not necessarily convex. However, experiments indicated that in many RoboCup scenarios, one constraint configuration is relevant - the penultimate and very simple case of Fig. 7.5, also shown in Fig. 7.6 (a). Here the resulting circular cuts can be approximated efficiently by multidimensional intervals, i.e., by box union constraints with two box elements or less. In the following, the propagation of line constraints is described. From a perceived line segment the robot can generate a constraint, i.e., a union of multiple hypercubes, cf. Fig. 7.7. The constraint consists of three-dimensional boxes; two dimensions are used for the position and one for the angle interval. The number of necessary hypercubes is determined by the number of lines on the field.

In this domain three different types of constraint propagation can occur – constraint propagation with:

- two circular constraints,
- a circular and a box union constraint,
- two box union constraints.

Fig. 7.6 depicts, how the different constraint types can be propagated. Besides flags and lines, also goal posts can be used. They result in circular constraints similar to flags. The two radiuses are defined by the measured distance to the posts and by their standard deviations.

7.3 Experiments

The algorithm was tested on three different platforms. The first platform is the simulator, the second platform is the Aibo four-legged robot. Final experiments were performed on the Nao two-legged robot. In the simulation the sensory data were not as noisy as on a real robot. Furthermore, the sensory data from distant lines could be perceived as well. Because of unlimited view distance, the experiments in the simulator could be executed with more perception data then on a real robot. With the help of Heinrich Mellmann, the following experiments were performed:

1. In the simulation we compared the running time of the constraint based localization approach with a Monte-Carlo particle filter. The robot was moving over the field.

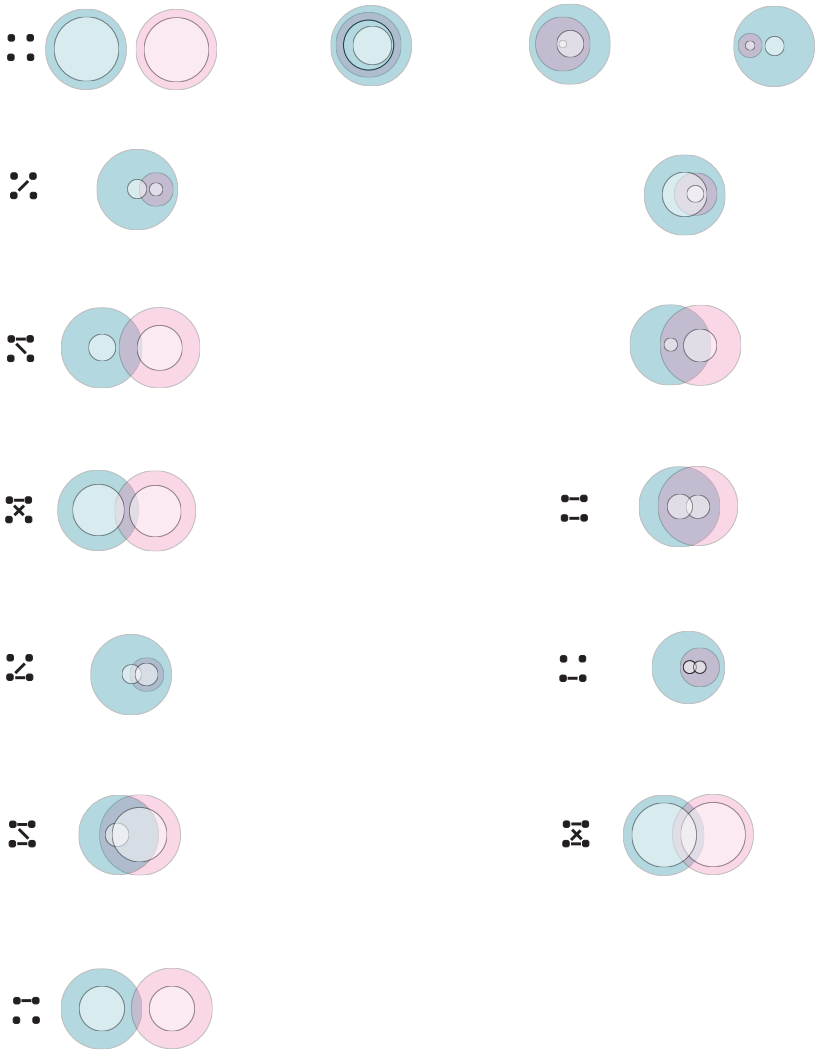


Figure 7.5: Constraint intersections generated by two circular constraints. The quadruple dots are encoding the intersections: Upper dots represent the outer radius, lower dots the inner radius of the constraints. Left or right dot positions stand for the left or right constraint. A line between dots denotes that the corresponding radiuses are intersecting.

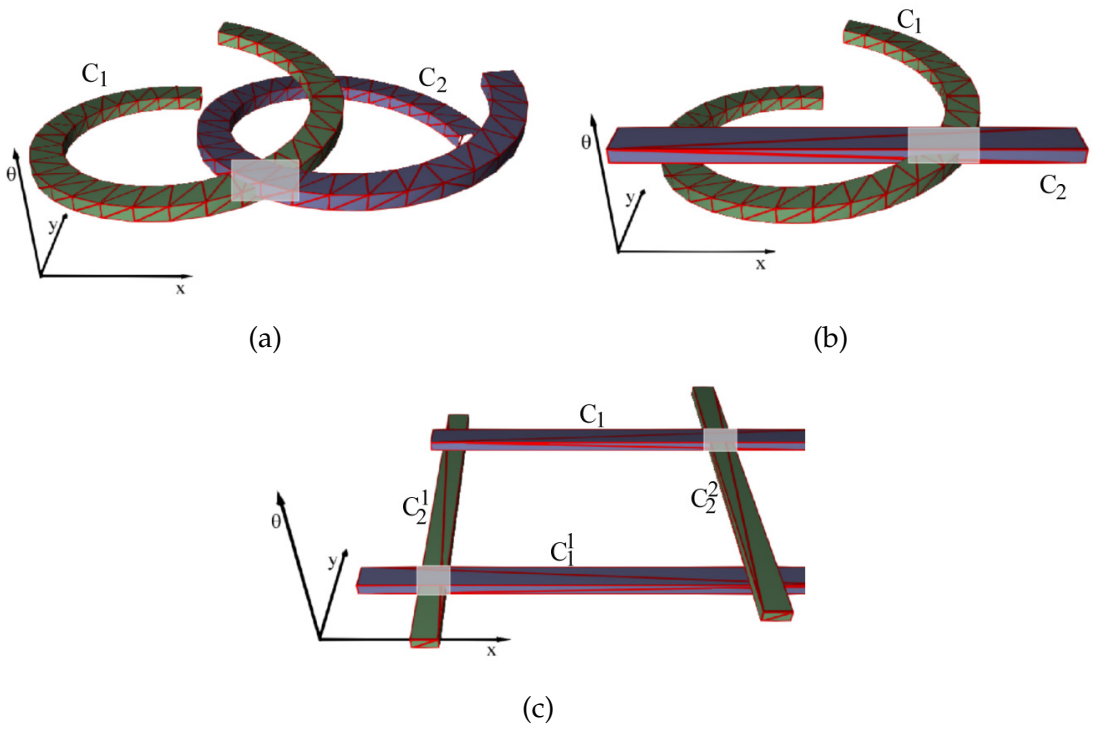


Figure 7.6: Constraint intersections visualized in 3-d space. Intersections are marked: (a) Two circular constraint are propagated. (b) A line constraint and a circular constraint, (c) two line constraints, each consisting of two boxes are propagated.

2. In a second experiment, the localization accuracy of the constraint based approach was compared to the Monte-Carlo approach. The robot was moving over the field.

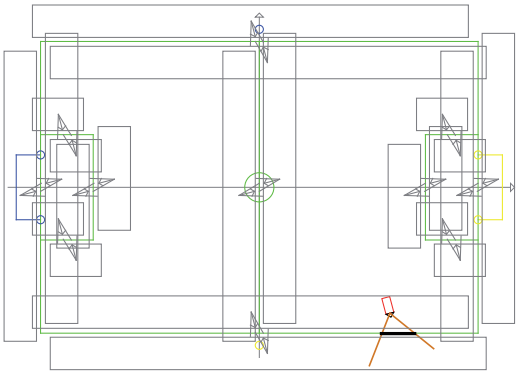
3. We tested the representational capabilities of both approaches in the simulator by using line percepts only. The robot was moving.

4. We tested the performance of soft-cuts as an alternative constraint propagation technique, especially the running time. The robot was moving.

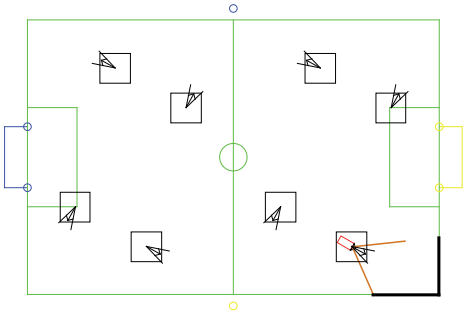
5. On the Aibo platform we tested the running time and the localization accuracy for a standing robot and compared it to the Monte-Carlo praticle filter.

6. On the Nao platform we again tested the localization accuracy for a standing robot.

7. We tested, how consistent and inconsistent sensory data constraints affect the localization accuracy through measuring the constraint ambiguity.



(a)



(b)

Figure 7.7: Constraints generated by line data and their propagation. The bold black lines depict the line segments the robot perceives. (a) Gray boxes describe the position belief of the robot, that was generated by a line percept. (b) Two constraints, generated by two line percepts are given. Black rectangles depict the resulting constraint, after intersecting the two line constraints.

7.3.1 Simulation

In the simulation we measured the running time, the localization accuracy, and the ability to represent certain beliefs.

Running time. In the first experiment the running time of the constraint localization approach was estimated within the simulator. As reference architecture served a Monte-Carlo Localization approach with 100 samples. The localization data was compared with ground truth measurements, the localization error was calculated and the running time was measured.

Time measurements in the simulator indicated that the constraint based localization was slightly quicker than a MCL approach, cf. Fig. 7.8.

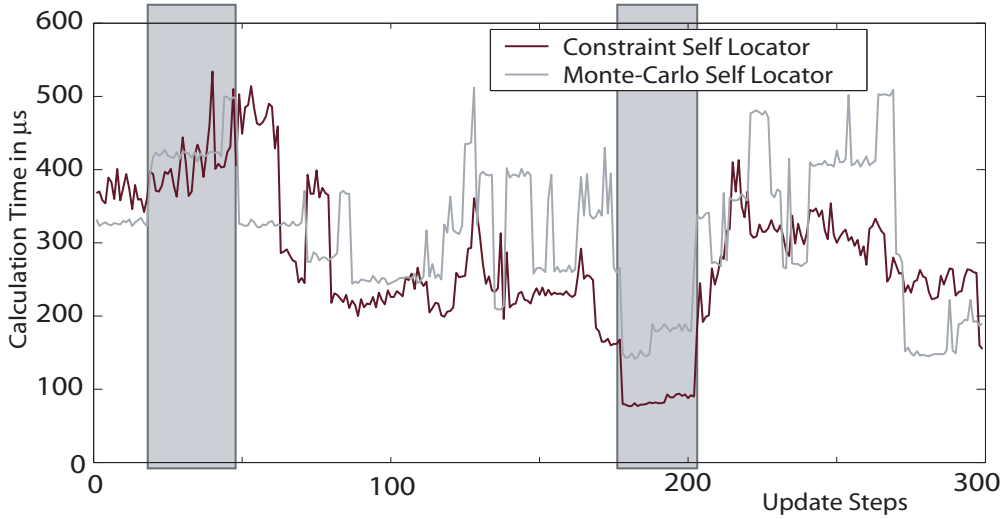
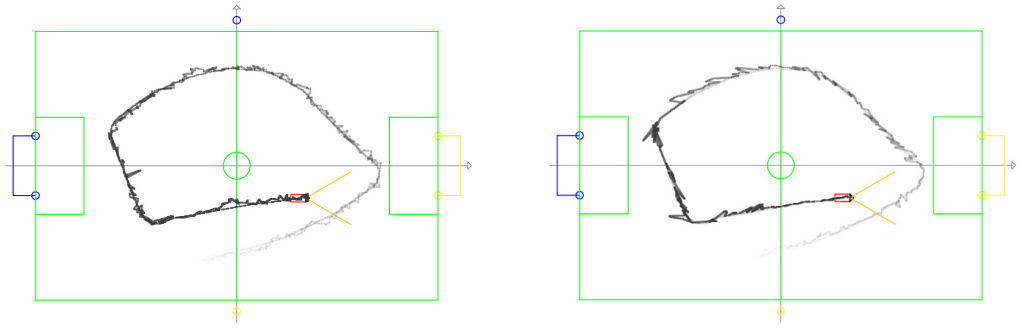


Figure 7.8: Runtime comparison, constraint- vs. particle-based approach on a 2 GHz Centrino processor. The left gray box emphasizes the time interval, in which 6 line percepts and two goal post percepts were available, the right box represents the calculation times when only one line percept was available. Gray plot: Monte-Carlo particle filter, using 100 particles. Average calculation time: 316μs. Dark plot: calculation time for a constraint based approach. Average calculation time: 279μs.

Localization Accuracy. In a second experiment the localization accuracy of both approaches was compared. Fig. 7.9 presents the modeled positions and the real robot position while the simulated robot was moving on the field. Fig. 7.10 gives information about the modeling accuracy of both approaches. Mostly, both localization approaches were comparably accurate. There were slight advantages for the constraint based approach when only lines were seen. The particle filter had problems to represent the whole probability distribution function correctly when the sensory data was very ambiguous. The constraint based localization showed to be more sensitive to noisy sensory data, which lead to little jumps in the modeled position.

Belief representation. In a third experiment, ambiguous sensory data was used to analyze the modeling accuracy in situations where only line sensory data is available, see Fig 7.7. In this case the constraint based approach showed a better representation ability of all possible robot positions. With a number of 100 particles large distributions could not be represented by the particle set. Comparable results could only be reached with a number of 500 particles. Especially when the robot was kidnapped and perceiving few sensory data, this was a limitation causing a higher relocalization time.



(a) (b)

Figure 7.9: Comparison, localization accuracy constraint vs. particle based. A robot runs a loop on the field (a) Monte-Carlo based localization, the straight line is the reference position, visible below the more oscillating line of the position estimate. (b) Constraint based localization.

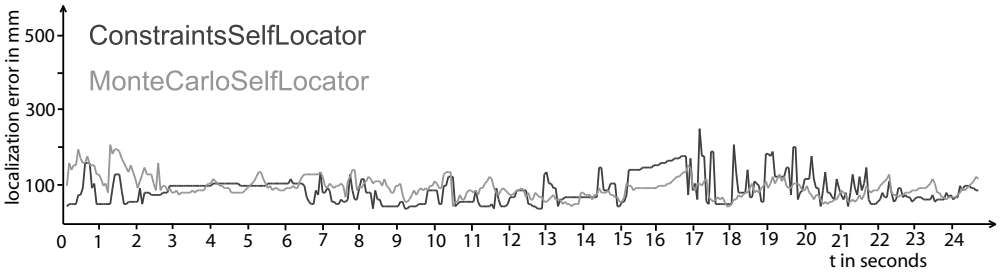


Figure 7.10: Localization error, constraint vs. particle based localization. Gray plot: Monte-Carlo Localization error, 100 particles used. Dark plot: localization error for a constraint based approach.

Soft-cuts. In another experiment the calculation time and accuracy of soft-cuts was measured. Therefore, a robot had to localize on the RoboCup soccer field again. We found out that soft-cuts can be computationally very expensive, especially when soft-cutting two box union constraints. Each constraint is generated from line percepts containing more than 20 box elements. The resulting constraint, which is generated from a soft-cut of two line constraints consists of more than 400 elements, see Fig 7.11 (b). To reduce the number of elements within each constraints we used a merge algorithm as described in Section 6.7.5. Here, constraints that resulted from merging the constraint elements became very large and moreover, contained large state space areas that did not belong to the original constraints, cf. Fig. 7.11 (c). Therefore, soft-cuts in the current form should not be used for intersecting box unions. But for propagating single box constraints soft-cuts lead to good results in terms of state approximation accuracy.

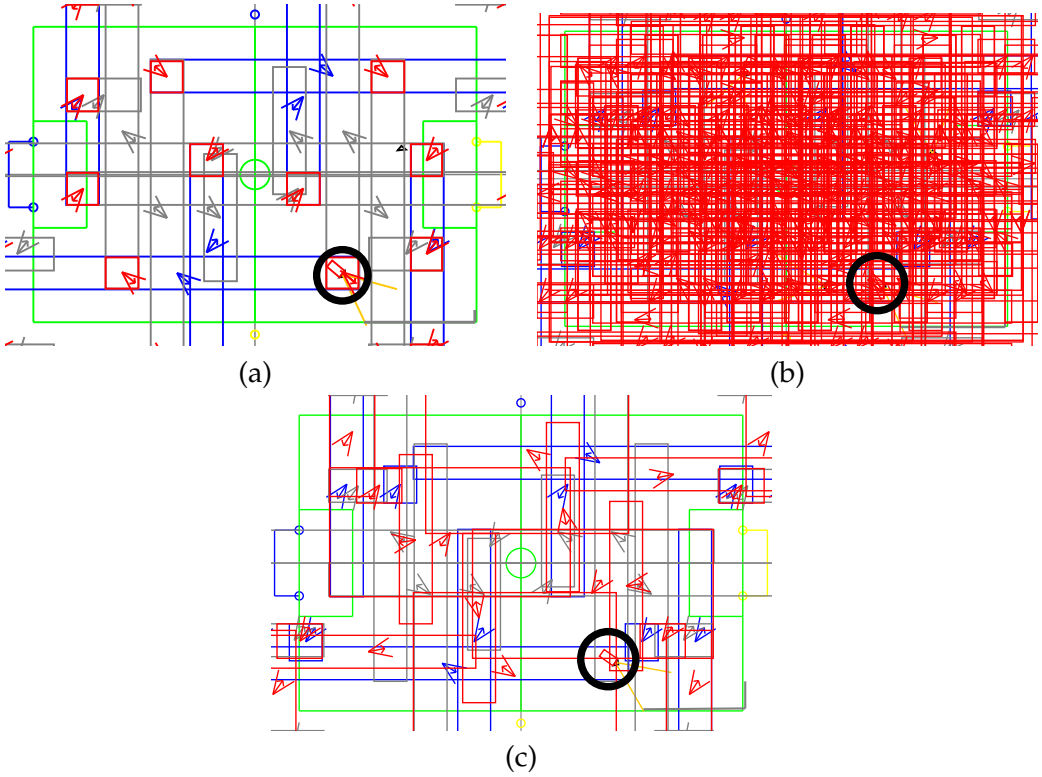
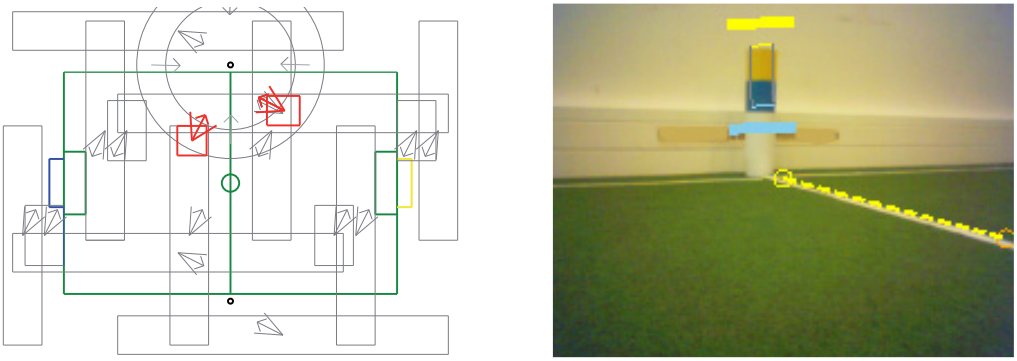


Figure 7.11: Comparison: constraint intersection vs. soft-cuts. Source constraints are gray and green, resulting constraints are red. The ground truth robot position is encircled. (a) Usual intersections, (b) soft-cuts between all constraint boxes, one can see that a huge amount of new box elements a created by the soft-cut, (c) soft-cuts with a maximum-distance heuristic.

7.3.2 Experiments on the Aibo Platform

As a first robot platform served the Aibo ERS-7. The limited camera resolution affected the ability to see distant field lines that were more than two meters away. Thus, therefore only few sensory data were available, cf. Fig. 7.12. The speed of the particle filter was improved by using lookup tables to incorporate line percepts [97]. Calculation times of both approaches were similar. It shall be pointed out that general comparisons are tricky, because there are many parameters and many possibilities for the implementation of Monte-Carlo particle filters. This comparison was made using the state of the art Monte-Carlo approach of the GermanTeam in 2007. Handling the noisy sensory data worked very satisfying on the Aibo, for both the particle-filter and the constraint based approach.



(a) (b)

Figure 7.12: Constraint localization experiment on the Aibo ERS-7: (a) Constraints, generated from scene (b): Recognized flag and line emphasized. The two emphasized red rectangles in (a) indicate that the sensory data from the image (b) leave two possibilities for positions on the field.

7.3.3 Experiments on the Nao Platform

The Nao two-legged robot served as a third platform to evaluate the constraint based localization algorithm. The main difference to the Aibo platform can be found in the lower walking stability, which is causing higher bearing angle errors, affecting the distance measurements to objects. On the other hand, image processing can be easier, as the robot's camera is mounted on a higher position (40 cm view height on the Nao compared to 10 cm view height on the Aibo).

On the Nao platform we analyzed how different constraints affect the localization accuracy, as presented in Fig. 7.13. Therefore we applied Alg. 6 from Chapter 6. The accuracy was measured as the area within 2-d space (x, y-position), that was covered by the belief constraint (upper row in Fig. 7.13). As percepts served the left and right goal posts and the field lines. Also was analyzed how consistent and inconsistent sensory data constraints affect the localization accuracy. It is visible that the area of the belief constraint decreased whenever consistent constraints were available, regardless of the number of inconsistent constraints. When sensory constraints were inconsistent or when no consistent constraints were available, the belief constraint became larger and the localization ambiguity increased. The figure presents the percepts as well, that the robot perceives when scanning the field from the center circle position. It also gives information about time intervals in which the robot does not perceive any landmarks.

Comparison of Localization Approaches. Even though we only implemented the Monte-Carlo Localization and the constraint based localization, we want to present a qualitative comparison of the constraint based localization approach, Kalman filters, EKF, UKF and the Monte-Carlo Localization.

As Table 7.1 shows, constraint based methods have an advantage regarding computational complexity compared to particle filters, especially when a large number of

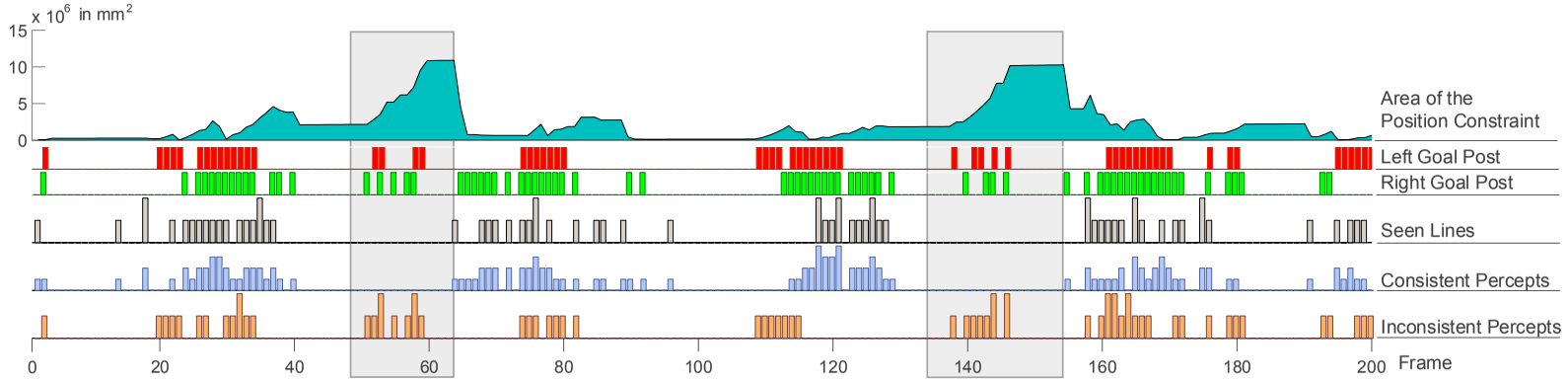


Figure 7.13: Constraint ambiguity over time, given different percept types. From the top to the bottom: (1) The area of the calculated position constraint as a measure of quality. (2),(3) indicate whether the left or right goal post were seen. (4) denote the number of seen lines. (5),(6) illustrate the overall number of perceptual constraints which were consistent or inconsistent with the position constraint, i.e., belief constraint. Gray boxes mark the areas when wrong goal percepts were available. During this time all seen percepts are inconsistent and the area of the position constraint grows until consistent data is perceived.

	KF/EKF,UKF	MHT	MCL	Constraint
Belief repr.	Gauss	Gss. Mix.	Part. distr.	Int. unions
Sens. data	Gauss/Arbitr.	Arbitr.	Arbitr.	Int. unions
Multimod.	-	+	+	+
Calcul.	+	o	-	o
Repr. Pow.	-	+	+	+
Amb. meas.	Cov. Matr.	Likelih.	Ent., Cov. M.	vol., max dst.

Table 7.1: Comparison of different localization methods: Kalman, EKF/UKF, MHT, MCL and Constraint based localization.

particles is necessary for representing the belief distribution. Kalman filters and their advanced versions EKF and UKF are computationally cheap but can only represent unimodal and Gaussian beliefs and thus, not solve the global localization problem. MHT can overcome this limitation. Regarding the sensory data, particle filters are most flexible, given a sufficient number of particles. Constraint based methods have a major advantage in the representational power.

Kalman filters, as well as EKF and UKF use covariance matrices to represent the modeling error. For MCL exist alternative measures to determine the particle convergence, e.g., particle distribution entropies or cross entropies. To determine the ambiguity of the belief in constraint based methods, one can use the constraint volume or the longest distance of elements within the constraint.

7.4 General Discussion

Using constraint based localization can be advantageous in terms of time complexity and for a variety of representation shapes. Constraints can be interpreted as binary distribution functions. This can be disadvantageous because all elements within the constraint have the same likelihood, on the other hand this proves to be a big advantage when it comes to representational power and ease of propagation. Storing widespread distributions takes only a few bytes. We defined a propagation method using simple-to-represent multidimensional conservative intervals. The performance of constraint based approaches depends on the number constraint parts within each constraints, which depends on the sensory data.

Future work. Some kinds of sensory data, that are usually hard to handle for Kalman approaches and Multi-Hypothesis Tracking is negative information, because of its multimodal sensor model, see Fig. 7.14. A localization approach taking advantage of negative information was introduced by Hoffmann et. al. [54, 53, 55]. The sensory data in this work was used to update a particle filter. Regarding constraints, the complex distribution functions of negative information can be approximated by a set of multidimensional conservative intervals, but which has not been done so far. Fig. 7.14 shows an example distribution for perceiving no flag or goal post, given that there are four

flags f_1, f_2, f_3, f_4 and two goals g_1, g_2 on the field. The probability density $p(z|x,y)$ tells, from which positions (x,y) a robot is likely to see no landmarks when looking straight ahead. The sensor model for negative sensory data from Fig. 7.14 was generated by integrating the three-dimensional sensor model $p(z|x,y,\theta)$ over all possible robot orientations θ:

$$p(z|x,y) = \int p(\underbrace{\neg z^{f_1}, \neg z^{f_2}, \neg z^{f_3}, \neg z^{f_4}, \neg z^{g_1}, \neg z^{g_2}}_{\triangleq z} | x,y,\theta) p(\theta) \mathrm{d}\theta \tag{7.2}$$

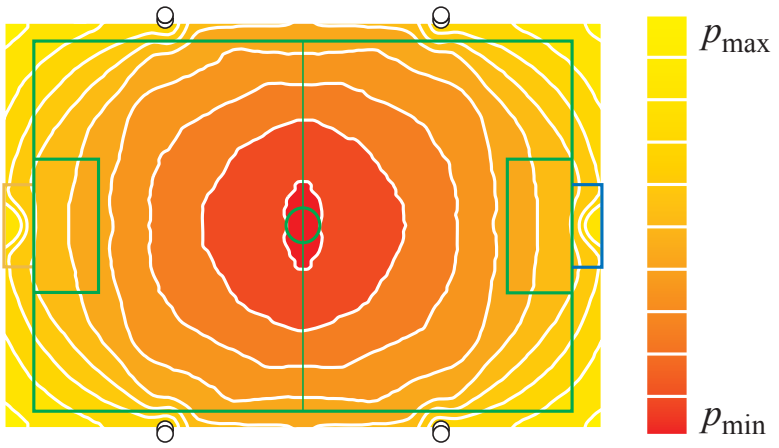


Figure 7.14: Negative information sensor model example. The probability for perceiving no flag or goal post given a certain (x,y) position on the field. For simplicity of the illustration, we did not consider robot orientations in the plot.

Using particle filters can be advantageous for those sensory data, because they do not make any assumptions about the distribution. But still, particle filters with a low number of samples will have problems to accurately represent the belief over the whole state space. In the next chapter the applicability of constraint based localization techniques for multi-agent modeling approaches will be analyzed.

8 Multiple Target Tracking

In the following chapter different possibilities of how to cooperatively track multiple objects are described and discussed. Multiple target tracking with a single agent has been widely analyzed within different domains. In [79], a set of leaves is tracked. A very famous application is the tracking of people, e.g., presented in [85]. A good description of Bayesian methods for multiple target tracking is given by [109]. In [28], a multiple target tracking approach using particle filters is introduced. When tracking multiple objects by a group of agents as in [116], the fusion of knowledge is usually necessary and in many cases useful, increasing the modeling accuracy, as described in [104]. In [61], a group of robots has to spread over an office floor area to be able to observe as much space as possible. An approach using robots to watch each other while moving, thus being able to reduce dead reckoning errors was presented in [94, 95]. In their work, a laser range finder was used to estimate the relative position to each other. In [100] theoretical aspects of two robots localizing each other are described for Kalman filters. Fox et al. presented in [11] how different robots can explore their surroundings, creating partial maps and fusing them. In [35] they showed, how robots equipped with cameras and laser range finders can collaboratively localize. A further known example of cooperation between robots is cooperative action planning. In many situations, the agents need to have a similar belief for a given situation. If a soccer playing robot wants to kick a pass to a team mate, it usually only succeeds when the team mate anticipates that action. Therefore, it is necessary that both players model their environment similarly.

But a model is not always necessary for cooperative behavior. Swarm robots, which imitate the behavior of ants, demonstrated that cooperation without direct communication between agents is possible, if there are common rules for all agents.

8.1 Data Association

Cooperative object modeling depends heavily on a reliable data association for all agents. Data association was used earlier, e.g., for radar tracking, when motions of different radar points had to be tracked over time. The situation in mobile robotics can be similar. One question is, how sensory data, that has been perceived at different times can be associated with the same objects, also known as the *correspondence problem*. Another question is, how can ambiguous object positions be associated with the same object track. Data association can be categorized into three groups [5]:

- Measurement-to-measurement-assignment: If there are multiple measurements of a scene, different features of a measurement should be assigned to features of

a second measurement.

- Measurement-to-path-assignment: In this case, multiple object tracks are available. Given a new measurement, the different observed objects have to be brought together with the available tracks. Later the tracks can be updated by the new measurements. This is also referred to as track maintenance or track updating.
- Path-to-path-assignment: Sometimes there are different models to one object. The association of different (partial) paths is the third category of data association.

Basically, there are two kinds of approaches to data association, as described in [5]. The first group mainly focusses on target objects. The goal is to find out how well the current sensory data matches the existing paths. The second group of approaches focusses on the sensory data, i.e., those approaches search for matching paths, and if none can be found, new paths are generated. Thus, the first group maintains existing paths, which is necessary when the measurements are very noisy, or when a single measurement is not sufficient to localize an object. The second group of approaches can be used to initialize new paths, whenever measurement data are very accurate and do not match with existing paths.

Furthermore, there are two different models on how to create a data assignment approach. The first model is called *deterministic*. The assumption is that the position of all objects is known from the beginning. In the next step, one calculates the most probable candidates for associating sensory data with the last known object positions. Of course, this association can be wrong. Subsequently, an approximation error is derived by calculating the likelihood of further assignment hypotheses.
The assignment result is then used in a standardized state estimation approach, e.g., in a Kalman filter or Extended Kalman filter. In the second approach, which is also called probabilistic model, the Bayes model is applied. Thereby, one calculates the probabilities for measurement X fitting object Y, which is then used for a state estimation algorithm.

In the following section, a different approach for data association is presented under consideration of different kinds of sensory data.

8.1.1 Different Kinds of Input Data and Object Models

Depending on the selectivity of the sensory data and the model of the objects to track, different approaches to data association can be used. The following models can be applicable:

- There are robot identifiers within the model and within the sensory data. Data association is trivial. This case is rare in reality, but can occur, e.g., when objects within the surroundings of the robot are equipped with RFID-chips.
- Robot identifiers are used within the model, but not within the sensory data. This case occurs relatively often, especially while tracking many target objects similar in appearance.

- There are no robot identifiers, neither in the model nor within the sensory data. This can occur when the number of objects to track is unclear. In reality this happens very often, e.g, when tracking cars or people. Sometimes it is not necessary to assign numbers to the objects, e.g., when one wants to know where there are free spaces only, regardless of which agent is present.

8.1.2 Correspondence Problem

Given two agents perceiving and communicating their percepts $s_1, \ldots, s_n$, to model two or more objects $r_1, \ldots, r_n$, cf. Fig. 8.1.

An easy approach is to use *Greedy*-association, in which n 2-tuples of sensory measurements and objects are created, so that the first percept s_1 is assigned to its nearest object r_i and the distance between them is calculated:

$$\text{dist}_{1,i} = |s_1 - r_i| \tag{8.1}$$

The distance is the difference between the position of the object r_i in the model and the percept coordinates s_1. Then the remaining sensory data and objects are iteratively associated with each other, until the sum of all distances reaches an upper bound. The remaining sensory data are not assigned to objects. Thus, noisy sensory data can complicate the data association and lead to more remaining data points. Another problem is that local minima can be found, i.e., data points being close to each other are associated, but as a result other pairs can have a larger distance between them and then the global optimum cannot be found, as shown in Fig. 8.2.

A method which finds the global minimum is called *least-squares-method*. It associates sensory data and modeled positions in such a way that the sum of squared distances is minimal:

$$\min \stackrel{!}{=} \sum_{i=1}^{N} \text{dist}(s_i, r_i)^2 \tag{8.2}$$

Therefore, it creates a table with distances of all (s_i, r_i) pairs and then searches for a combination of all entries, so that every element occurs in one pair only, and the number of associations is maximal. Of course the calculation can become computationally expensive, due to its exponential complexity, given the number of sensory data. The greedy method, in contrast, has linear complexity.

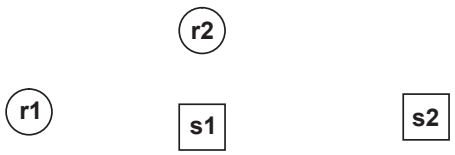


Figure 8.1: The correspondence problem: Two agent communicate sensory data s_1, s_2 of objects r_1, r_2. The correspondence problem is how to combine sensory data with different objects.

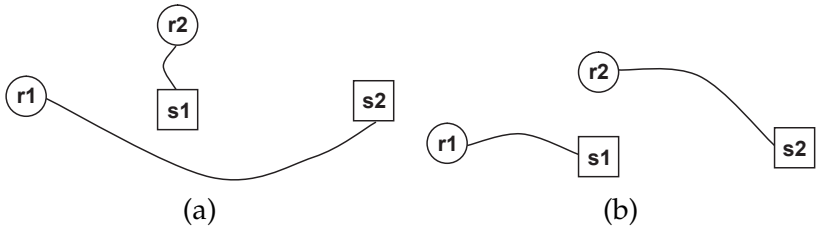


Figure 8.2: Data association methods. (a) Greedy-methods can lead to local minima and thus, fail to find the global minima. (b) The least squares method always finds the global optimum.

Even though the least-squares method achieves practicable results in terms of assigning the correct percept data to the modeled objects, the application of a Joint-Probabilistic-Data-Association-Filter, (abbrv. *JPDAF*)[20], can improve the results [25, 105] when distribution functions are given.

8.1.3 Joint Probabilistic Data Association Filters

The following theoretic description of JPDAFs was described earlier in the work of Schulz et al. in [105, 107]. Assuming, n objects have to be tracked, where $X^t = \{x_1^t, \ldots, x_n^t\}$ describes the state of the objects at time t. Every x_i^t is a probabilistic variable defined over the state space of a single object. Furthermore, let $Z^t = \{z_1^t, \ldots, z_{m_t}^t\}$ be the vector of sensory measurements at time t. Z^t is a sequence of all measurements until time t. As described earlier, the task is to assign the measurements to the different objects.

A *joint association event* ψ describes a set of pairs $(j, i) \in \{0, \ldots, m_t\} \times \{1, \ldots, N\}$. The association function ψ determines which measurement, e.g., percept was assigned to an object. If an object could not be assigned, we write z_0^t. All possible joint association events are denoted by Ψ_{ji}, i.e., events which assign object i to measurement j. At time t the JPDAF describes the posterior probability of assigning measurement j to object i by:

$$\beta_{ji} = \sum_{\psi \in \Psi_{ji}} P(\psi | Z^t). \tag{8.3}$$

The individual probabilities $P(\psi|Z^t)$ can be calculated as follows:

$$\begin{aligned} P(\psi|Z^t) &= P(\psi|Z(t), Z^{t-1}) \\ &\overset{Markov!}{=} P(\psi|Z^t, X^t) \\ &\overset{Bayes!}{=} \alpha p(Z^t|\psi, X^t) P(\psi|X^t), \end{aligned} \tag{8.4}$$

where α serves as a normalizing constant. The term $P(\psi|X^t)$ corresponds to the probability for the association ψ given object X^t. In some situations it can be useful to assume

$P(\psi|X^t)$ for every object to be equal, so that the term can be approximated by a constant [105, 107].

$p(Z(t)|\psi, X^t)$ describes the probability of a measurement Z^t, assuming the objects are in state X^t and the specific association ψ is given. To calculate this probability, the possibility of a "false positive", i.e., the wrong detection of an object, must be considered. If the number of false positives is given by $(m_t - |\psi|)$, then $\gamma^{(m_t-|\psi|)}$ is the probability of all "false positives" in Z^t given ψ. Assuming all other measurements to be independent of each other, the following holds:

$$p(Z(t)|\psi, X^t) = \gamma^{(m_t-|\psi|)} \prod_{(j,i)\in\psi} p(z_j(t)|x_i^t). \tag{8.5}$$

In case of a Kalman based object tracking, the terms $p(z_j(t)|x_i^t)$ are represented by normal distributions. But for the general case of equation (8.4) we have:

$$P(\psi|Z^t) = \alpha\gamma^{(m_t-|\psi|)} \prod_{(j,i)\in\psi} p(z_j(t)|x_i^t). \tag{8.6}$$

Since $P(\psi|X^t)$ can be seen as a constant, as described, the term can be represented by α. JPDAFs use for state update $p(x_i^t)$ the recursive Bayesian filtering rule with the prediction:

$$p(x_i^t|Z^{t-1}) = \int p(x_i^t|x_i^{t-1}, n)p(x_i^{t-1}|Z^{t-1})\mathrm{d}x_i^{t-1} \tag{8.7}$$

and correction step:

$$p(x_i^t|Z^t) = \alpha p(Z(t)|x_i^t)p(x_i^t|Z^{t-1}) \tag{8.8}$$

In most cases it is unclear which sensory measurement from $Z(t)$ was caused by object i, therefore $p(x_i^t|Z^t)$ can also be calculated using the joint association probability β_{ji}.

$$p(x_i^t|Z^t) = \alpha \sum_{j=0}^{m_t} \beta_{ji} p(z_j^t|x_i^t)p(x_i^t). \tag{8.9}$$

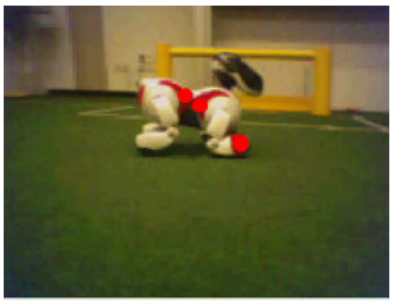
Consequently, only the models $p(x_i^t|x_i^{t-1}, n)$ and $p(z_j(t)|x_i^t)$ have to be known. Those models again depend on the tracked object positions and on the used sensors.

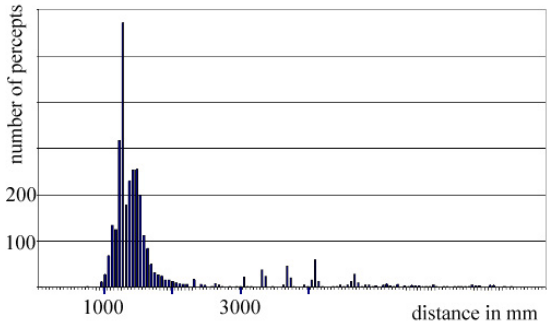
Using the described approach enables a more robust and more accurate data association for cooperative object modeling, even though this approach can be expensive in calculational needs [78]. Thus, for the here described robot tracking approach we decided to use a greedy data association method.

8.2 Model Fusion within the Standard Platform League

Now we want to introduce a possibility of tracking multiple robots, that cannot be distinguished from each other. In our robot code we mainly performed bearing based distance measurements to the robot within the image. The distance to the objects is es-

timated using the lowest point within the image assuming this point is on the ground. The measurement distribution is presented in Fig. 8.3 (b). The distance measurement is very erroneous, whereas the angle measurement is more accurate, so an observing robot is able to estimate the direction to another robot precisely, but the distance is more difficult to estimate. Moreover, it is challenging to distinguish different robots standing next to each other. Therefore, in the GermanTeam code from 2004 no agents but occupied areas on the field were modeled [52]. Modeling was done within a two





(a) (b)

Figure 8.3: Measured distance distribution for a player percept: (a) Player percepts, depicted by the red dots and generated using the red color of the robot. (b) The diagram shows the number of measured distances within a certain time, given a robot standing at a distance of 1m.

dimensional grid representing the field. When a player R_1 perceives another player R_2, the perceiving player can model the position of the perceived player using an egocentric model and then project this model into field coordinates, using its self-localization distribution function, or using percept-relations.

Transformation from Egocentric into Allocentric Coordinates

Transformation of egocentric distribution functions e into allocentric distributions a can be described mathematically as the convolution of e with the localization distribution function s of the robot pose on the field. In the one-dimensional case the resulting distribution function a of the observed robot or object position on the field can be calculated as:

$$a(x) = e(x) * s(x) = \int e(x-\tau)s(\tau)\mathrm{d}\tau \tag{8.10}$$

A one-dimensional example for the propagation of different object distribution functions with the robot position distribution function is given in Fig. 8.4. In the two dimensional case, one has to perform a two-dimensional convolution to get the resulting allocentric distribution a of an object from the robot pose s and the egocentric object

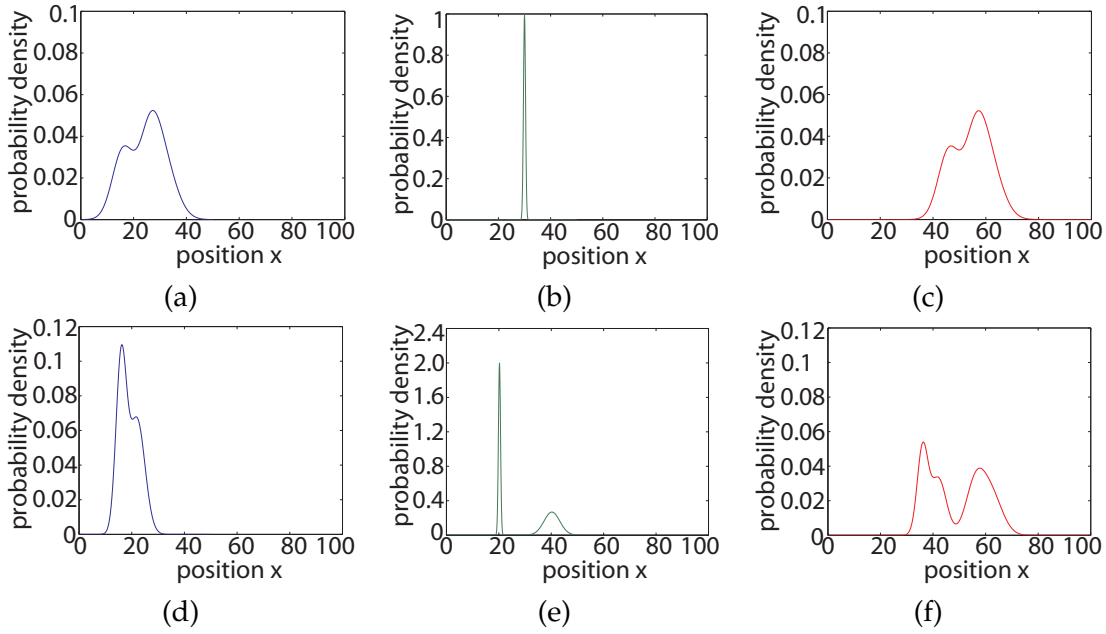


Figure 8.4: Egocentric to allocentric coordinate transformation 1-d: (a) and (d) show probability density functions for seeing a robot within a certain distance. (b) and (e) present distributions for the robot on the field; (c) and (f) present resulting distribution functions of the observed robot on the field.

model e:

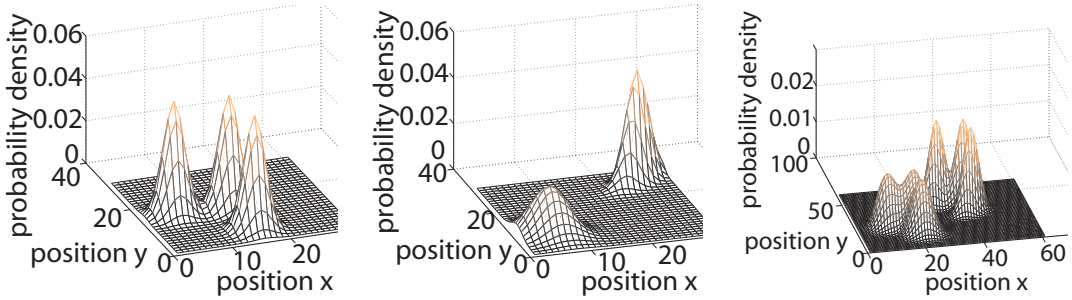
$$a(x,y) = e(x,y) * s(x,y) = \int\int e(x-\tau, y-\kappa)s(\tau,\kappa)\mathrm{d}\tau\mathrm{d}\kappa \tag{8.11}$$

Fig. 8.5 shows an example of a transformation of an egocentric distribution into allocentric coordinates, cf. Fig. 8.5 (c), using the egocentric distribution of the object to model (a) and the robot pose distribution (b). Fig. 8.5 (c) demonstrates that the uncertainty within robot localization affects the allocentric object distribution. In a last step the rotation distribution of the robot should be considered as well. Rotational transformations are non-linear operations with regard to the x, y position of the object to model. To make calculations easier, it is assumed that the rotational distribution $s(\theta)$ is independent from the x, y self-localization distribution, cf. Fig. 8.6. In a first step the egocentric object position e is transformed into e'_θ using the rotational distribution function of the robot $s(\theta)$:

$$e'_\theta(x,y) = \int e(x\cos(\theta) - y\sin(\theta), x\sin(\theta) + y\cos(\theta)) \cdot s(\theta)\mathrm{d}\theta \tag{8.12}$$

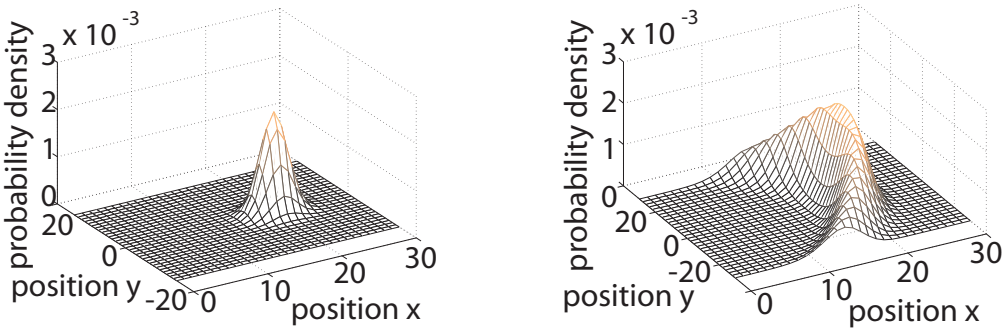
This function e'_θ can then be used in equation (8.11) to transform the translational distribution of the object to model into allocentric coordinates.

A more general equation for arbitrary distributions can be derived by using the transformation equation (4.1) for egocentric to allocentric models from Chapter 4, where a



(a) (b) (c)

Figure 8.5: Egocentric to allocentric coordinate transformation 2-d: (a) Example distribution for an egocentric object model, (b) robot position distribution, no rotation component, (c) resulting allocentric object distribution after transformation.



(a) (b)

Figure 8.6: Egocentric to allocentric coordinate transformation 2-d, angle only: (a) Egocentric model, (b) allocentric model after transformation, robot has a high variance within its orientation θ.

stood for the allocentric position, r for the robot position and e for the egocentrically modeled object, e.g., a ball or a robot:

$$T_{r_x,r_y,r_\theta}(e_x,e_y,e_\theta) \triangleq \begin{bmatrix} a_x \\ a_y \\ a_\theta \end{bmatrix} = \begin{bmatrix} \cos(r_\theta) & \sin(r_\theta) & 0 \\ -\sin(r_\theta) & \cos(r_\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} + \begin{bmatrix} r_x \\ r_y \\ r_\theta \end{bmatrix}$$

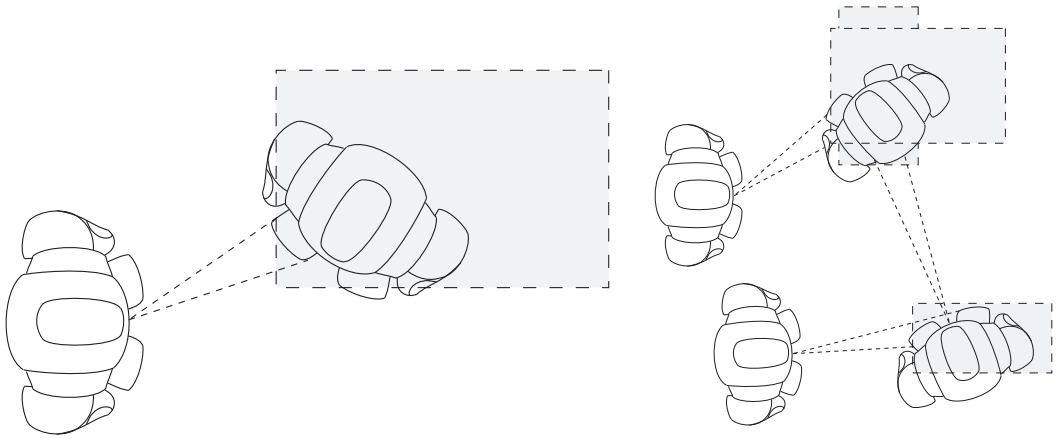
Thus, the allocentric distribution for the egocentrically modeled object can be calculated as:

$$p(a_x,a_y,a_\theta) = \int\int\int \underbrace{p(r_x,r_y,r_\theta)}_{\textbf{robot pos. distr.}}\, p(\, \underbrace{T_{r_x,r_y,r_\theta}(e_x,e_y,e_\theta)}_{\textbf{cond. alloc. object distr.}} \,|r_x,r_y,r_\theta)\mathrm{d}r_x\mathrm{d}r_y\mathrm{d}r_\theta \qquad (8.13)$$

This equation (8.13) can be interpreted as the Bayes formula, where the resulting allocentric distribution is derived by integrating the conditional allocentric distribution (given a robot position) over the robot position distribution. T serves as the transformation function. Abstracting from the different coordinates one gets:

$$p(a) = \int p(r) \cdot p(T(e)|r) \mathrm{d}x \tag{8.14}$$

The calculation of the distribution becomes inefficient with increasing dimensions. Already for three dimensions, the calculation becomes too complex for an accurate approximation. Thus, for an online running implementation, optimizations such as *Maximum Likelihood Estimation* or other approaches have to be used. At this point it should suffice to have described, how such transformation works theoretically.



(a) (b)

Figure 8.7: Generating constraints from player percepts: (a) A robot is perceiving another one and generating a position constraint in egocentric or allocentric coordinates. (b) A group of robots is perceiving different agents. Different belief constraints are generated. If those belief constraints are modeled in allocentric coordinates and communicated throughout the group, every agent can generate a model containing the information from all robots.

8.3 Constraint Based Single-Agent Multiple-Object Tracking

At first an implementation shall be given, where a single robot keeps a model of a group of other robots. Therefore, the robot has to maintain a set of beliefs to its robot percepts. The frame of reference can be egocentric or allocentric. Fig. 8.7 (a) illustrates how an egocentric player percept constraint is generated. If no communication is necessary and if the self-localization error is high and odometry data is available, an egocentric model can be useful. If the agents communicate (cf. Fig. 8.7 (b)), an allocentric model

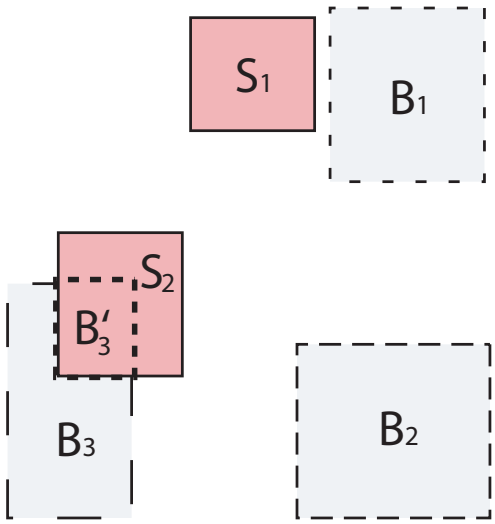


Figure 8.8: Constraint based player modeling, data association. Each robot percept must be assigned to a robot belief, if robot beliefs exist. A distance criterion is used for data association. If the distance between all robot beliefs and the robot percept is too large, a new belief is created - if the maximum number of beliefs is reached so far, the closest belief to the robot percept position is set to the percept position. In this example B_1 is set to S_1, S_2 is intersected with B_3, resulting in a new $B_3^{'}$.

is to prefer. The following data association approach can be applied for egocentric and allocentric models.

Constraint data association. Whenever a robot is tracking another robot, the sensory data constraint has to be assigned to a robot model, i.e., the sensory data has to be used for updating a robot position constraint. The modeling process distinguishes the following situations:

- Without any prior knowledge, the new sensory data constraint will generate a new belief constraint for a robot.
- If there are robots modeled already and the new sensory data constraint is consistent with at least one robot position constraint, the sensory data constraint is used for updating (Fig. 8.8). If there is more than one candidate, it is used for updating the most likely candidate, i.e., the most consistent one.
- If the sensory data constraint is inconsistent with all robot position constraints, a new robot position is generated, but only if the maximum number of robot position constraints has not been reached, otherwise:
- If the maximum number of robot constraints has already been reached, the new

data is applied to the least inconsistent robot position constraints, in the easiest way by resetting the nearest constraint to the new sensory data. Soft-cuts and other techniques as the *Inconsistency Data Rate* for model resetting of the closest robot position constraint would be possible as well.

The algorithm, presented in Alg. 7 was implemented and tested. Fig. 8.9 illustrates how the algorithm works in a soccer field scenario. At this stage it is possible that different player percepts are assigned to the same player model. The approach can be changed in a way that only one percept is used for correcting a player model, and the next player percept has to be used for the nearest other belief, or if the maximum number has not been reached, the percept is used to create a new belief constraint representing a robot position.

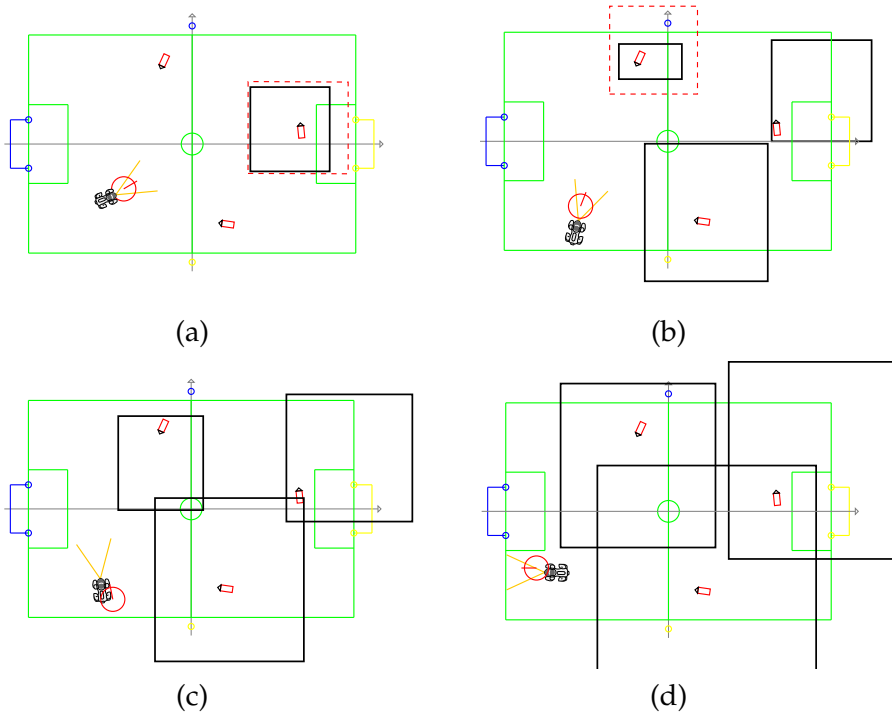


Figure 8.9: Single-agent player modeling. The observing robot is depicted as an Aibo-robot. The circle depicts the estimated own position. Three red rectangles depict the robots to track. (a) The observing robot has created one model and later (b) three models of all three robots and is looking at the upper one. The dotted red square depicts the robot percept constraint, the three black squares represent the belief of the three robot positions. (c) and (d): The observing robot is perceiving not any robot, the uncertainty about the observed robot positions increases (the constraint borders increase).

Algorithm 7: Single-agent multi-robot tracking

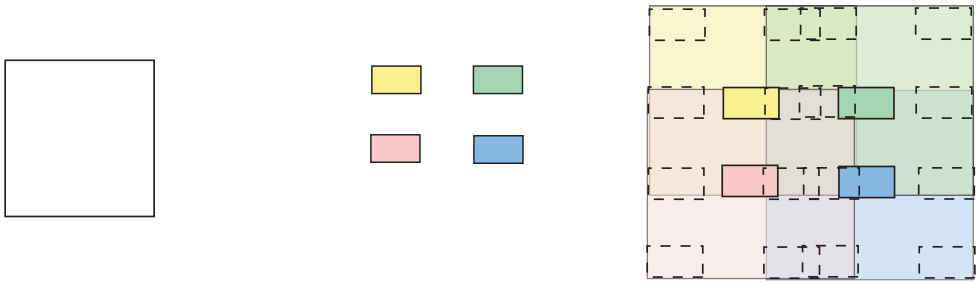
Input: $B_{t-1} = \{B^{(i)}|1,\ldots,n\}$ representing the belief about the robot positions, observation set Z_t, self-localization belief x_t

```
1  B_t := B_{t-1}                                          //Initialization
2  for z_t^(r) ∈ Z_t do
3      a_t^(r) := transform(z_t^(r), x_t)                  // egoc. percepts to alloc. coords.
4      d_r := generateDistances(a_t^(r), B_{t-1})          // calculate distances to all
                                                           // robot position models
5      if ((m := d_r.getMinElement()) < θ) OR (card(B_t) = MAX_MODELS) then
6          update(B^(m), a_t^c)                            // update the closest robot model if // the closest
           robot position model is nearer than a threshold or // if the maximum number of
           models has been reached
7      end
8      else
9          if card(B_t) < MAX_MODELS then
10             B_t := B_t ∪ generateNewModel(a_t^(r)) // generate new model
11         end
12     end
13 end
14 return B_t
```

8.4 Constraint Based Multi-Agent Multiple-Object Tracking

When tracking multiple, non-distinguishable targets, it is hard to assign the sensory data correctly to the targets. There are two kinds of sensory data available. When the perceiving robot is localized, it can transform the egocentrically modeled sensory data to global ones. The second possibility is to use percept-relations between the tracked object and fixed reference objects. For transforming egocentric constraints into global ones, one has to use the Bayes formula (8.14) from Section 8.2. An example transformation for egocentric to allocentric constraints is presented in Fig. 8.10. Since the player recognition is not very accurate for humanoid robots and in combination with self-localization uncertainty, allocentric player models often come with a high degree of uncertainty.

Local player model and team player model. A cooperative player recognition was implemented as follows: Each robot models the positions of other robots in allocentric coordinates, resulting in a *local player model* of the other players, cf. Fig. 8.11. It is called *local*, because no communication is necessary for its creation. In the implementation, for simplicity reasons and ease of calculation, the probability distribution function of each modeled robot position is approximated by a single box constraint instead of box union constraints. Consequently, each robot has to maintain a set of constraints for the



(a) (b) (c)

Figure 8.10: Egocentric to allocentric constraint transformation: (a) Robot position constraint, (b) egocentrically modeled player constraint, (c) resulting player constraint after transformation to field coordinates, using Bayes law.

set of robots to observe, in the experiments there were four robot positions to maintain. Each robot is then communicating the modeled positions about itself and about the other players to all the other robots. Each robot resets a second model, the *team player model* to the values of its local player model. The team player model is used to assign all the communicated player data from the other robots, as shown in Fig. 8.11. To decide, which constraints have to be associated with each other, a simple distance measure is used, similarly to the sensory data association of the modeled constraints (see Fig. 8.8). When a robot is receiving player model data from n other robots, it has to perform the data association process $n \cdot n$ times, i.e., for each incoming data set, n robot positions have to be assigned. In the implementation the data association of the communicated data was done using weighted soft-cuts. Calculating soft-cuts for two singe box constraints are computationally cheap. Using weights for the input constraints of the soft-cut makes the soft-cut operation associative and solves the ordering question, i.e., which of the constraints are propagated first, second, etc. For example, having three robots A, B, C, the first two robots A, B could soft-cut their model with a weight of 0.5, resulting in the $A, B - belief$. The resulting constraint is derived by averaging the constraint vectors of the source constraints. The model of the third robot C is weighted with one third and the $A, B - belief$ with two thirds, cf. Fig. 8.12, which leads to the same result as for different soft-cut orders.

8.5 Experiments

Because the player recognition accuracy was low on the real robot platforms at the time the experiments were performed, experiments were conducted in the simulator.

Local model experiments. The setup was as follows: The observing robot is localizing, using the constraint based localization approach and transforming the egocentric player percepts into allocentric coordinates. The player models were created in allo-

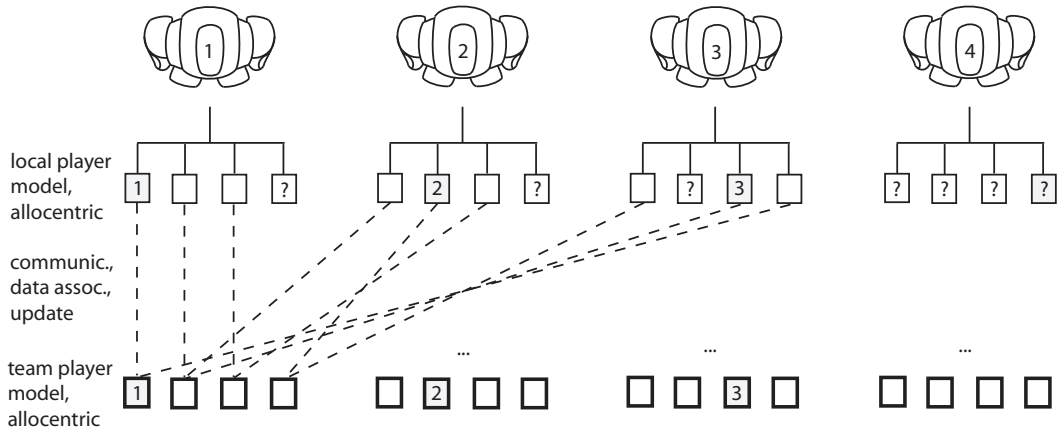


Figure 8.11: Team-player model scheme

centric coordinates as well, as described in Alg. 7. The following experiments were performed:

1. At first was analyzed, how the sensory update rate, i.e., the frequency by which other robots are seen, affects the model accuracy. Therefore, the observing robot rotated on its position, resulting in frequently incoming sensory data of robots within its surroundings. The other three robots to observe were moving arbitrarily on the field. They were modeled, using the data assignment algorithm described in Section 8.3.

2. In a second experiment the observing robot was orientated towards the robots to observe. Thus, the perception gaps were reduced.

3. In the last experiment the player percept error was increased.

The results of the first experiments have been plotted in Fig. 8.13 (a) and (b). The data indicate, as expected, that a low player data update rate results in increasing modeling errors. In some cases, e.g., for robot number 3, the localization error is decreasing at modeling step 130. This is because sometimes the robots are coincidentally moving back to the positions at which they have been seen for the last time. The accumulated modeling error, i.e., the sum of the modeling error of all three robots was between 150cm and 250cm.
In the second experiment, the observing robot was looking more often at the other robots, so there were less perception gaps compared to the first experiment. The results of the second experiment are presented in Fig. 8.13 (c) and (d). Because of the higher update rate, we expected smaller modeling errors of the player model. In the beginning the model error was high, because other players had to be found for the first time. After modeling step 70, the accumulated modeling error decreased to 70cm till 170cm.

In the third experiment, the error of the player percepts was increased. As a result, the accumulated modeling error, i.e., the modeling error sum of all three modeling errors

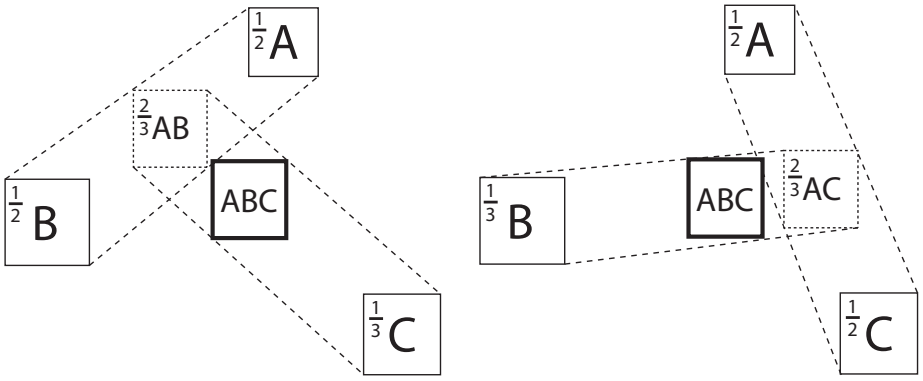


Figure 8.12: Soft-cut order invariants: When using weighted soft-cuts, the soft-cut becomes associative, i.e., the order of soft-cutting different constraints is not important. (a) Constraint A was soft-cut with constraint B (A and B were weighted with 0.5), resulting in AB. AB was then soft-cut with C, the weights were two thirds for AB and one third for C, leading to constraint ABC. (b) A different soft-cut order is leading to the same constraint (ABC).

increased. It is notably high for some modeling steps, when the third player has not been seen for a certain time. During the rest of the time, the modeling error increased only slightly, compared to the second experiment.

Concluding, data association of percept data to the player models worked accurately. The observing robot accurately updated the three player models with the player percepts. Furthermore, we saw that in a dynamic scenario where the robots are moving, it is very important to receive percept data about the modeled players without big time gaps. This is because in times, where no player percepts are incoming, it is hard to predict the motion of the modeled robots.

Team model experiments. Now the team model data association and model accuracy was tested. All experiments were performed in the simulator. We did not consider communication delay in the model, because the resulting position error is very small compared to the other errors as from self-localization and local player model, and so far the robots were moving slowly. The experimental scenarios are:

1. In the first scenario data association for the team model was tested. Thereby it was interesting to see, if all robots acquire the same model about their team-mates. All the robots were moving.

2. The same scenario as before, but with higher self-localization and local player model error.

3. Now the errors of the team model and the local model were compared for low update frequencies (5 percent, i.e., each robot was visible in 5 percent of the images).

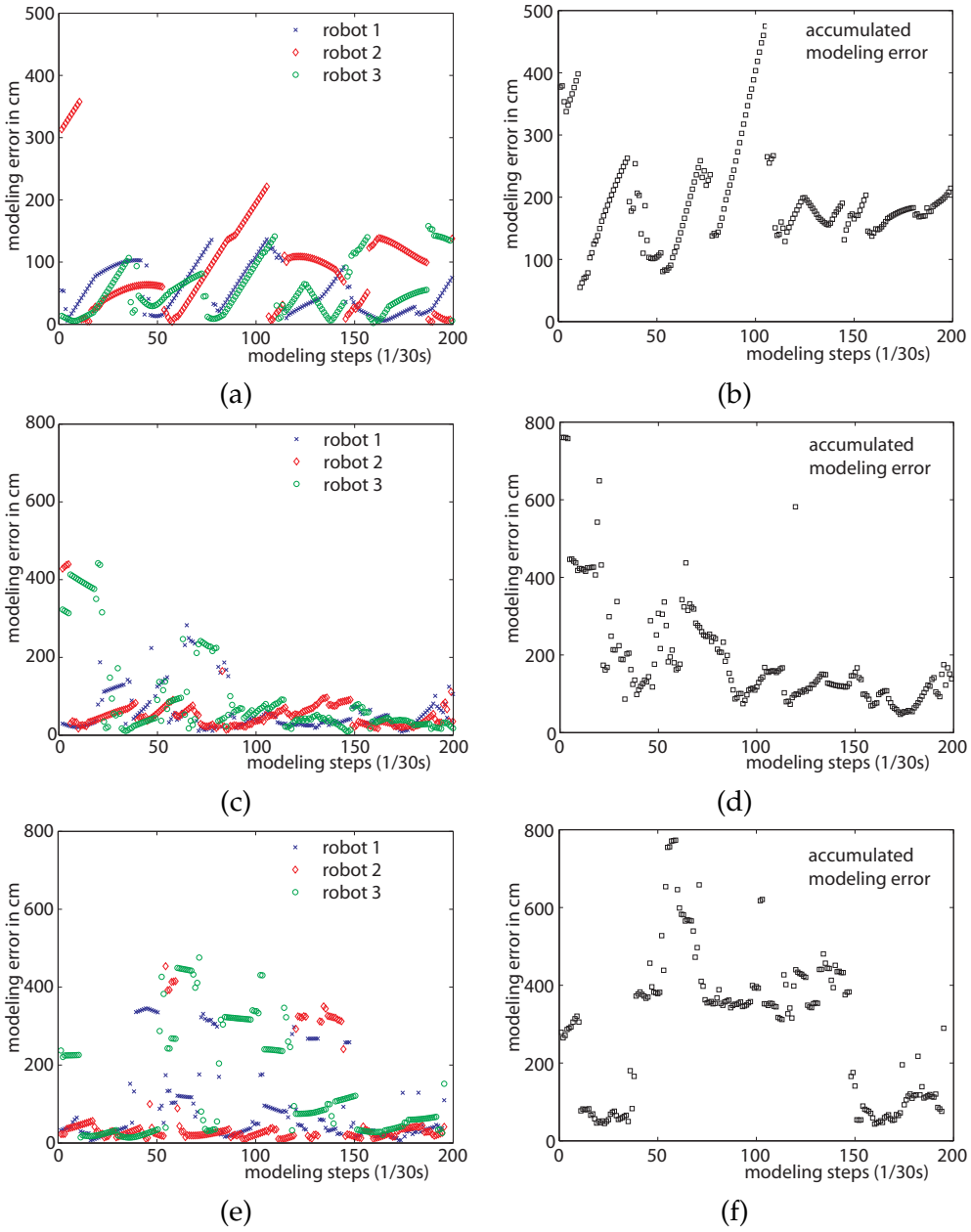


Figure 8.13: Player modeling error experiments, three robots to track. The diagrams show the difference between the modeled positions and the real positions for all three robots (left column) and the sum of the three errors (right column). Self-localization errors: $\sigma_{sl} = 20cm$, playerpercept error: $\sigma_{ppc} = 30cm$. (a) and (b): The observing robot is receiving data from surrounding players with a low frequency. (c) and (d): Higher player data update frequency. (e) and (f): Same player data update frequency as in (c), but higher percept error, $\sigma_{ppc} = 60cm$.

All the robots were moving.

4. The experiment is performed again, but for higher update frequencies (20 percent).

5. Were compare the modeling errors of a static with a dynamic scenario, i.e., in one setup all the robots are standing, in the other setup the robots are moving.

6. In the last experiments we compared the modeling error for higher self-localization and local player model errors. The update frequencies were 5 percent and 100 percent.

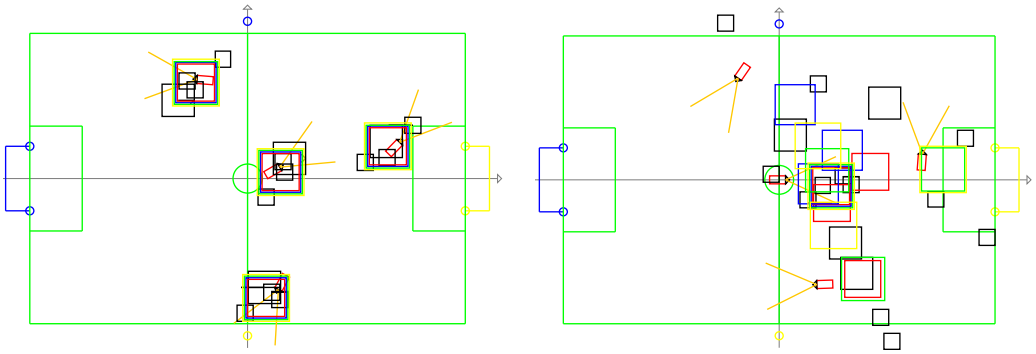
In the setup of the first experiment all four agents were moving with constant speed in the simulator and turning from time to time, to stay on the field. Self-localization was possible, the localization error was for both dimensions x and y: $\sigma_{sl} = 30cm$; the error of the modeled and observed robots was $\sigma_{plr} = 25cm$ for each dimension. Fig. 8.14 shows the data association result. If the distance between the robots is not too small (Fig. 8.14 (a)), and the modeling error is as described, the team models of the different robots are equivalent, i.e., each robot has the same belief about the positions of the four robots. In experiment 2 the modeling errors were doubled, and the distance between the robots became smaller, cf. Fig. 8.14 (b). Now the team models of the different robots differed. This was caused by the data association order, which can have an effect on the outcome of the model when many association candidates are equally close.

In experiment 3 the error of the local model was compared to the error of the team model. Therefore, the distance of a real robot A to the closest constraint position, i.e., to the next best fitting constraint, that another robot B has, was measured for the local and the team model - and compared, see Fig. 8.15. We found out that the team model was in both cases ((a) and (c)) more accurate than the local model, i.e., which was generated without communication. In case of low update rates, the error of the team model was $98cm$, compared to $125cm$ for the local model. For higher update rates in experiment 4 (20 percent), the error in both models was lower, as expected. The team model error was $37cm$, the local error was $28cm$.

In the experiment 5, the team model error in dynamic situations was compared to the error in static situations. As expected, the error in static situations was much lower, see Fig. 8.16 (a) and (b). In experiment 6 the team modeling error was tested again for higher localization errors, see Fig. 8.16 (c) and (d). Thereby the update frequency was adjusted to 5 percent and in another run the robots had permanent updates (100 percent). The error in case of permanent updates was clearly smaller and remained constantly low.

8.6 Summary

Creating and maintaining a multiple-object model can be a very complex task. In some cases, the robots to track cannot be distinguished from each other. Useful metrics, such as the distance of the percept coordinates to the model coordinates have to be applied.



(a) (b)

Figure 8.14: Multi-agent player modeling - data association result. Team models of different robots are depicted by different colors. (a) Big distances between the robots, small localization errors: $\sigma_{sl} = 30cm$ for self-loc., $\sigma_{plr} = 25cm$ for local player model. The team models of different robots are similar; (b) smaller distances between the robots, modeling errors doubled: $\sigma_{sl} = 60cm$, $\sigma_{plr} = 50cm$.

Another question is which frame of reference to apply. Egocentric models are useful when self-localization is inaccurate, allocentric models are necessary whenever the modeled positions shall be communicated to other robots. Furthermore, there is a variety of data, that can be incorporated. Percept relations can be helpful when an object model is updated by a group of agents. Disadvantages are the widespread belief distributions which need more processing power to be incorporated into the model than simple distributions. In this chapter was described how constraint based approaches can be used for multi-agent multiple-object tracking tasks. Under some assumptions they can be applied using little processing power. Constraint based data association and model transformation from egocentric to allocentric coordinates and model combination by different agents has been successfully demonstrated and implemented.

However, it shall be pointed out that constraint based methods can handle noisy sensory data to a certain extent only. If the sensory data and models get too inaccurate, the constraint borders have to be enlarged and the model becomes more ambiguous. This limitation is similar to other approaches such as Kalman filters which are robust to sensory noise to a certain extent only. The advantages of constraint based approaches are their efficiency regarding model transformation from egocentric to allocentric coordinates, their ability to represent and to incorporate percept-relations and last but not least the possibility of being easily communicated from one robot to another. Concluding, the benefit of communicating player models in many cases depends on the ability to create allocentric models – unless robots can localize objects relative to other robots. An accurate self-localization is a keystone for the creation of allocentric models, even though percept-relations can be useful as well when self-localization is erroneous. The second question is, how frequently the observing robots perceive other players. If in-

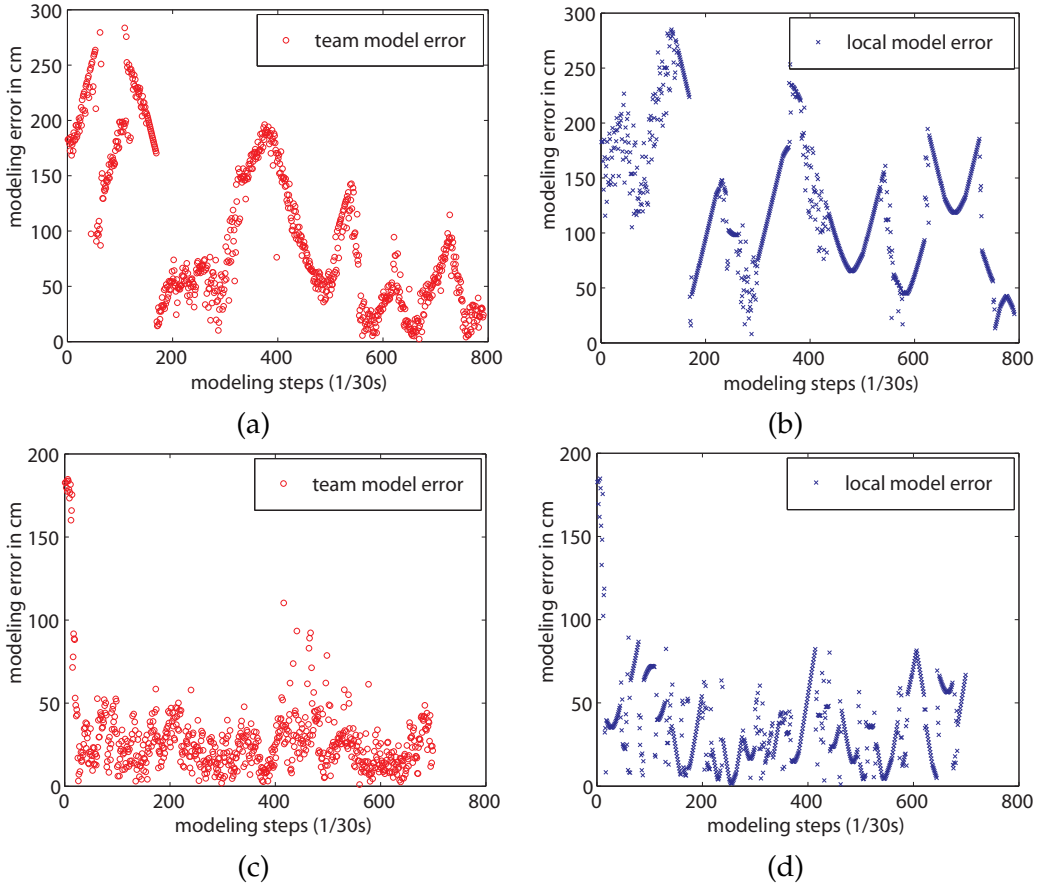


Figure 8.15: Local vs. team model errors. The modeling errors in the plots are errors for one reference robot position to one robot position in the model. Localization errors in both cases: $\sigma_{sl} = 30cm$ for self-loc., $\sigma_{plr} = 25cm$ for local player model. In (a) and (b), the update frequency was 5 percent, i.e., a robot was visible to another agent in 5 percent of all images; the team model had an error of $\sigma_{team} = 98cm$, the local model of $\sigma_{local} = 125cm$. (c) and (d): Update frequency 20 percent: $\sigma_{team} = 28cm$, $\sigma_{local} = 37cm$.

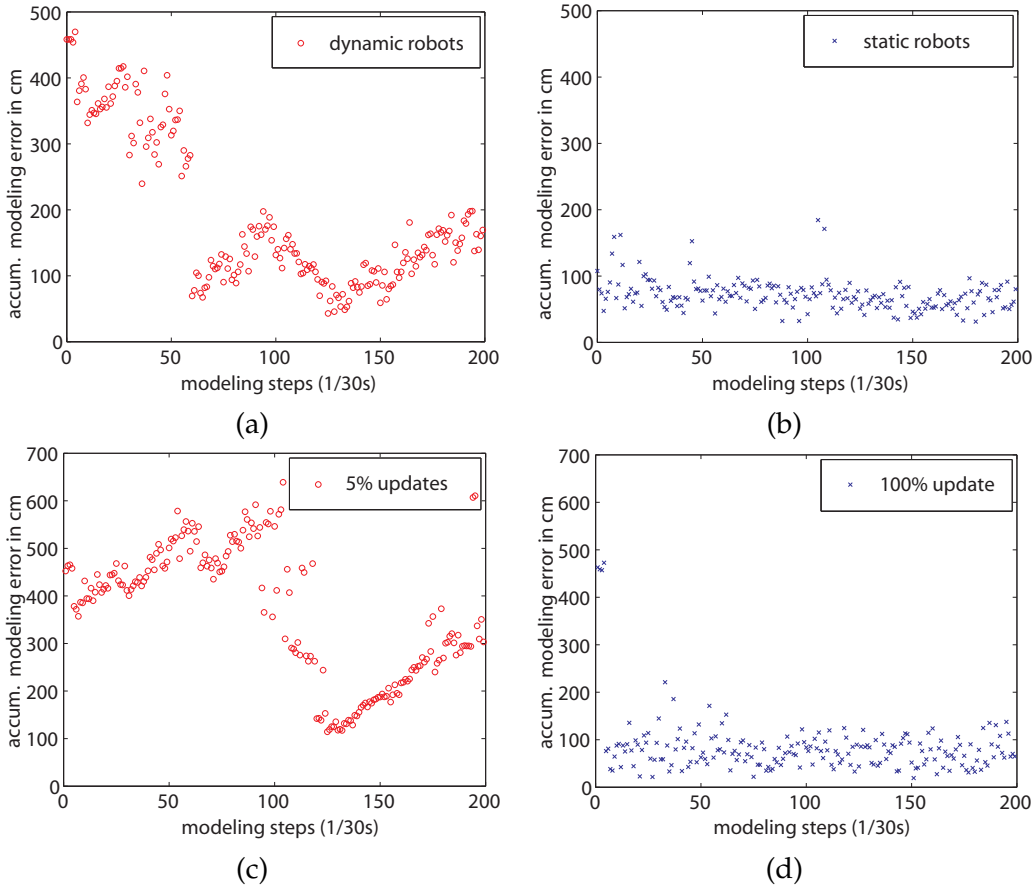


Figure 8.16: Multi-agent player modeling - static vs. dynamic scene: Localization errors were: $\sigma_{sl} = 60cm$ for self-loc., $\sigma_{plr} = 50cm$ for local player model. The team modeling errors in the plots are cumulated errors for all four modeled robots. (a) and (b): Comparison of team model errors in a static and a dynamic scene. (c) and (d): Again, the model errors were measured, this time for low update frequencies and for permanent updates.

coming sensory data of other players is rare, communication between agents is more useful than in cases where each robot has sufficient evidence about the positions of the modeled players.

9 Conclusions and Future Work

Object modeling and self-localization consist of state estimation and perception. Some of the biggest challenges for perception are partial observability [69] and noisy sensory data. Thus, the key questions asked in this work were:

"Can world modeling be improved when a group of agents is interchanging information about its surroundings?" and *"Can constraint based modeling techniques support a group of agents to maintain a computationally efficient model of their environment?"*

The author of this work beliefs that the answers to these questions depend on the availability of useful resources to perceive.

In Chapter 4 was analyzed which sensory data are useful for single-agent and for cooperative object modeling. We demonstrated the sensory error-correlation of percepts within the same image. Consequently, the concept of percept-relations was introduced as a possibility to object modeling whenever self-localization information is noisy or not available. Percept-relations can easily be communicated to other robots which predestines them as an input data for multi-agent object modeling. It was described and shown experimentally, how cooperation between agents in form of sensory data communication enables robots to improve their object model accuracy and also their self-localization accuracy.

Nevertheless, the success of this approach highly depends on the availability of multiple percepts or features within an image and on the availability of reference objects within the environment. Furthermore, in this approach all participating agents have to have a consistent map about their surroundings.

In Chapter 5 was presented how percept-relations in combination with non-linear regression can be used to calculate the speed of moving objects within an allocentric frame of reference, under some assumptions about the motion properties of the object to track and about the communication accuracy. It was also shown that the accuracy of the object speed estimation can be affected by communication delays. Thus, a simple algorithm was presented, that can be applied to setups with small communication delays. In the last part of this chapter was discussed how the calculation of a cooperative model can be distributed within different robots of the group in terms of robustness to network failures and calculational efficiency.

In Chapter 6 basic concepts for a constraint based world model were presented. In robotics, sensory data is usually noisy, so the generated constraints do not necessarily have to be globally consistent, thus, they do not have to contain a global solution. But there is a solution in reality. Also there can be a variety of solutions that cannot be constrained further. We introduced two measures, one for inconsistency and one for ambiguity of the constraint set and analyzed mathematical properties of the given functions. As a strategy to find a solution for a given constraint set, minimal conservative

intervals were introduced in form of n-dimensional intervals. They possess some useful properties as seclusiveness under the intersection operator and they can be represented efficiently.

After the theoretical part, in Chapter 7, a constraint based localization approach was implemented for self-localization. Therefore, an algorithm using three-dimensional box union constraints was implemented. Different strategies to handle inconsistencies were tested. We showed in simulation and on real robots that constraint based localization can be slightly quicker and as accurate as Monte-Carlo self-localization. However, the effectiveness of the given approach hinges again on the kind of percepts within the robot's environment. It was demonstrated that some sensory data cannot be used as effective as others, i.e., some sensory data produce simple shaped constraints, as circles or boxes, and others generate more complex ones.

In Chapter 8, a constraint based multi-target tracking algorithm was introduced. In the first part was presented how a single agent can keep track of multiple objects, i.e., players on the field while representing its belief as a set of belief constraints. Later, a group of agents had to create a constraint based team model of all its members. Therefore the constraints could be communicated between the robots.

In the experimental part was demonstrated that the team model, which was created through communication of the local models, was less erroneous in terms of the positioning error than the local models, which were created without communication. This was tested for different update frequencies.

Of course, the accuracy of the sensory data and the frequency by which new sensory data are perceived, affects the accuracy of the constraint based localization. If sensory data is too inaccurate, the constraint based localization and the local model become inaccurate as well or no state estimation is possible at all. Concluding, the main contributions of this thesis are:

- Introduction of spatial relations between objects and demonstration of different algorithms, which use percept-relations to cooperatively localize objects, estimate their speed and to improve self-localization.

- Communication principles and its effects on modeling accuracy were discussed, an approach to handling communication delays within the modeling process was presented.

- A constraint based modeling technique, that enables a robot to represent its belief about the environment was presented. Different strategies to find and to handle inconsistent, i.e., erroneous sensory data were discussed, tested and compared to other modeling techniques.

- The author presented how constraint based modeling approaches can be applied to multi-agent multi-object tracking. Therefore was analyzed, how multi-modal egocentric distributions can be transformed into allocentric distributions, to be communicated to other agents.

- Data association techniques for multiple objects tracking by a group of robots were discussed. One of those techniques was implemented and tested in the simulator.

9.1 Future Work

Future work to improve the presented approaches can be done on:

Motion control based on resource availability. This was proposed earlier in [69]. The main idea is to let a robot adjust its motion speed by the certainty of its localization or tracking model. If the robot is uncertain about its position and surrounding objects, e.g., because of highly inconsistent sensory data, it should move slower until it gathers an accurate model. If so, the robot can continue moving faster as long as the model stays accurate. This is similar to human behavior, humans tend to be extra perceptive and to walk cautious in crowded streets to avoid running into other people.

Fuzzy Constraints. The key idea is that constraints are no longer modeled as sets containing multidimensional elements with degrees of membership of 1. Instead constraints could be modeled as Fuzzy sets, where each element has a degree of membership between 1 and 0. Thereby inconsistencies of two constraints would not result in an empty intersection set, but maybe in a Fuzzy set with low degrees of memberships. To analyze analogies of this model to probabilistic approaches would be an interesting task as well.

Learning strategies for constraint parameters. Learning the constraint parameters can help to improve the model accuracy. For example the robot could learn the best ratio of inconsistent to consistent constraints with regard to modeling accuracy, and adjust the thresholds for when to reject sensory data constraints and when to reject a belief constraint which does not match the sensory data at all.

Learning active sensing strategies. In different environments there are different landmarks which can be perceived with differing sensing accuracies. To learn, which landmarks are most useful for constraint based localization would help to automate the tuning of sensing strategies.

Motion prediction. Modeling the motion of passive objects has been widely analyzed. Another work in progress is to anticipate the motions of an agent, which can be highly non-linear.

Belief representation of constraints. In this work was presented how many distributions can be approximated by box union constraints or by circular constraints. For some sensory data as negative information or bearing-only percepts, the resulting position

distribution becomes more complex, thus, other constraint shapes might provide better approximations for those distributions regarding calculational efficiency and minimality of the conservative constraints.

Robot pose and perception constraints. The data of the camera image hinges on the joint angles of the neck and the legs. Inconsistent object sizes can help to correct these joint angles, which always have a small position error. Some work on this topic was done in [81], but without conservative interval constraints. It will be interesting to see how constraint based approaches can be applied to those problems.

Numerical approaches. Instead of using the propagation of conservative intervals to find a global solution of a given constraint set, it will be interesting to analyze how a solution can be found numerically, e.g., by maximum likelihood methods or more complex optimization approaches with related algorithms.

9.2 Summary

Thanks to scientific advances, which result in cheaper robots with more abilities, the field of mobile robotics shifted from singe-agent to multi-agent systems where the robots have to cooperate to perform certain tasks. This thesis has demonstrated that cooperative creation and maintenance of a world model by multiple agents is possible in real-time. The keys to this success are methods which allow the robots to handle and to reason about percepts of uncertainty. Furthermore, this thesis presented methods that enable robots to handle constraints within their surroundings and described how robots can cooperatively create a world model within a constraint framework. Finally, it could be demonstrated that cooperation between agents can lead to a more accurate world model.

Although the presented algorithms were applied to mobile robots with visual sensors, they can be used within other domains which require real-time perception as well.

Robot Platforms

.1 Sony Aibo ERS-7

The first robot platform used for experiments is the Aibo ERS-7, manufactured by Sony, see Fig. .1. The advantages of this platform are its robustness, stability, reliability and its far developed software. Since this robot has four legs it is very stable. One bottleneck comes from the camera, which is only 10 cm above the ground.

Category	ERS-7
Resolution Camera	$208 \cdot 160$
Frame rate (fps)	30
Opening angle vertical	44 deg.
Opening angle horizontal	55
CPU - clockrate (MHz)	576
RAM (MByte)	64

Table .1: Hardware specification ERS-7

Figure .1: Sony Aibo ERS-7.

This, combined with a limited resolution makes it uneasy to accurately detect certain features as field lines and distances to objects on the field in general. If distances are long, objects are very close to the horizon and the vertical bearing based distance measurements become erroneous. More specification data is presented in Table .1.

.2 Aldebaran Nao

The second platform used in this work is the French robot "Nao" from Aldebaran, see Fig. .2. It is a humanoid robot, about 60 cm high. In its first version, that was used here it has one VGA camera enabling the robot to perceive images in a reasonably high resolution.

Category	Nao
Resolution Camera	$640 \cdot 480$
Frame rate (fps)	30
Opening angle vertical	45 deg.
Opening angle horizontal	36 deg.
CPU - clockrate (MHz)	600
RAM (MByte)	256

Table .2: Hardware specification Aldebaran Nao

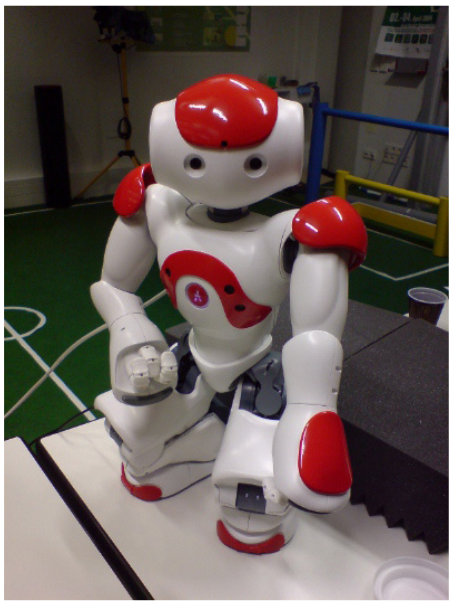


Figure .2: Aldebaran Nao.

Compared to the Aibo, the robot is slower and less stable when walking. Since the robot is swinging a lot while walking, distance and angle measurements can be erroneous. At least while standing, because of the higher position of the camera, distances to objects can be calculated with a higher accuracy than on the Aibo platform. More specification data is presented in Table .2.

Bibliography

[1] K.O. Arras and S.J. Vestli. Hybrid, high-precision localization for the mail distributing mobile robot system. In *Proceedings of the IEEE International Conference on Robotics & Automation*, 1998.

[2] K.O. Arras, J.A. Castellanos, and R. Siegwart. Feature-based multi-hypothesis localization and tracking for mobile robots using geometric constraints. In *Proc. of the IEEE International Conference on Robotics & Automation*, 2002.

[3] D. Austin and P. Jensfelt. Using multiple gaussian hypotheses to represent probability distributions for mobile robot localization. In *Proc. of the IEEE International Conference on Robotics & Automation*, pages 1036 – 1041, 2000.

[4] N. Ayache and O. D. Faugeras. Maintaining representations of the environment of a mobile robot. In *IEEE Transactions on Robotics and Automation*, volume 5, pages 804–819, 1989.

[5] Yaakov Bar-Shalom, X. Rong Li, and Thiagalingam Kirubarajan. *Estimation with Applications to Tracking and Navigation*. Wiley-Interscience, 2001.

[6] G. A. Borges, M. j. Aldon, and T. Gil. An optimal pose estimator for map-based mobile robot dynamic localization: Experimental comparison with the EKF. In *In IEEE International Conference on Robotics and Automation*, 2001.

[7] F. Bourgault, T. Furukawa, and H.F. Durrant-Whyte. Process model, constraints, and the coordinated search strategy. In *Proceedings of the 2004 IEEE Int. Conf. on Robotics and Automation (ICRA'04)*, pages 5256–5261, New Orleans, LA, April 2004.

[8] W. Burgard, A.B. Cremers, Dieter Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and Sebastian Thrun. The interactive museum tour-guide robot. In *Proc. of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998. Outstanding paper award.

[9] Wolfram Burgard, Dieter Fox, Daniel Hennig, and Timo Schmidt. Estimating the absolute position of a mobile robot using position probability grids. In *In Proceedings of the Thirteenth National Conference on Artificial Intelligence, Menlo Park*, pages 896–901. AAAI, AAAI Press/MIT Press, 1996.

[10] Wolfram Burgard, Dieter Fox, and Daniel Hennig. Fast grid-based position tracking for mobile robots. In *Lecture Notes in Computer Science*, KI-97: Advances in Artificial Intelligence. Springer Berlin / Heidelberg, 1997.

[11] Wolfram Burgard, Dieter Fox, Reid Simmons, and Sebastian Thrun. Collaborative multi-robot exploration mark. In *Proceedings of the IEEE Int. Conf. on Robotics and Automation*, pages 476–481, 2000.

[12] Wolfram Burgard, Christopher Baker, Zachary Omohundro, Scott Thayer, and William Whittaker. A system for volumetric robotic mapping of abandoned mines. In *In Proc. IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[13] Anthony R. Cassandra, Leslie Pack Kaelbling, and James A. Kurien. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In *In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 963–972, 1996.

[14] J. A. Castellanos, R. Martínez-cantín, J. D. Tardós, and J. Neira. Robocentric map joining: Improving the consistency of EKF-SLAM. In *Robotics and Autonomous Systems*, volume 55, pages 21–29, 2007.

[15] Jose A. Castellanos, J. M. M. Montiel, J. Neira, and J. D. Tardós. The SPmap: A probabilistic framework for simultaneous localization and map building. *IEEE Transactions on Robotics and Automation*, 15:948–953, 1999.

[16] R. Chatila and J.P. Laumond. Position referencing and consistent world modeling for mobile robots. *Proceedings of the IEEE International Conference on Robotics and Automation, St. Louis*, pages pp. 138 – 145., 1985.

[17] R. Cheeseman and P. Smith. On the representation and estimation of spatial uncertainty. *International Journal of Robotics 5*, pages 56 – 68, 1986.

[18] Howie Choset and Keiji Nagatani. Topological simultaneous localization and mapping (SLAM): Toward exact localization without explicit localization. *IEEE Transactions on Robotics and Automation*, 17:125–137, 2001.

[19] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley Series in Telecommunications, Wiley, New York, 1991.

[20] I.J. Cox. A review of statistical data association techniques for motion correspondence. *International Journal of Computer Vision*, 10(1):53–66, 1993.

[21] Ernest Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32, 1987.

[22] M.C. Deans. Maximally informative statistics for localization and mapping. In *Proc. of the IEEE International Conference on Robotics & Automation*, 2002.

[23] Frank Dellaert, Wolfram Burgard, Dieter Fox, and Sebastian Thrun. Using the condensation algorithm for robust, vision-based mobile robot localization. In *1999 Conference on Computer Vision and Pattern Recognition (CVPR '99)*, June 1999.

[24] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte carlo localization for mobile robots. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1322–1328. IEEE, 1999.

[25] M. Dietl, J. Gutmann, and B. Nebel. Cooperative sensing in dynamic environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'01), Maui, Hawaii*, 2001.

[26] M.W.M. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 17(3), 2001.

[27] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and Computing*, 10:197–208, 2000.

[28] Arnaud Doucet, Ba-Ngu Vo, Christophe Andrieu, and Manuel Davy. Particle filtering for multi-target tracking and sensor management. In *In Intl Conf. Information Fusion*, pages 474–481, 2002.

[29] H.F. Durrant-Whyte. Uncertain geometry in robotics. *IEEE Transactions on Robotics and Automation 4*, 1:23 – 31, 1988.

[30] Y.C. Eldar, A. Beck, and M. Teboulle. A Minimax Chebyshev Estimator for Bounded ErrorEstimation. In *IEEE transactions on signal processing*, volume 56 of *IEEE Transactions on Signal Processing*, pages 1388–1397, April 2008.

[31] Alexander Ferrein, Lutz Hermanns, and Gerhard Lakemeyer. Comparing sensor fusion techniques for ball position estimation. In Ansgar Bredenfeld, Adam Jacoff, Itsuki Noda, and Yasutake Takahashi, editors, *RoboCup 2005: Robot Soccer World Cup IX*, Lecture Notes in Artificial Intelligence, pages 154–165. Springer, 2006.

[32] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte carlo localization: Efficient position estimation for mobile robots. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence (AAAI)*, pages 343–349. The AAAI Press/The MIT Press, 1999.

[33] Dieter Fox. Adapting the Sample Size in Particle Filters through KLD-Sampling. *International Journal of Robotics Research*, 22:2003, 2003.

[34] Dieter Fox, Wolfram Burgard, Sebastian Thrun, and Armin B. Cremers. Position estimation for mobile robots in dynamic environments. In *In Proc. of the National Conference on Artificial Intelligence (AAAI)*, 1998.

[35] Dieter Fox, Wolfram Burgard, Hannes Kruppa, and Sebastian Thrun. A probabilistic approach to collaborative multi-robot localization. *Auton. Robots*, 8(3): 325–344, 2000. ISSN 0929-5593.

[36] Dieter Fox, Jeffrey Hightower, Lin Liao, Dirk Schulz, and Gaetano Borriello. Bayesian filtering for location estimation. *PERVASIVE computing*, pages 10–19, July–September 2003.

[37] Udo Freese. *An O(log n) Algorithm for Simultaneous Localization and Mapping of Mobile Robots in Indoor Environments*. PhD thesis, Universität Erlangen Nürnberg, 2004.

[38] A. Gelb. *Applied Optimal Estimation*. MIT Press, 1974.

[39] Zoubin Ghahramani. An introduction to hidden markov models and bayesian networks. In H. Bunke and T. Caelli, editors, *Hidden Markov Models: Applications in Computer Vision*, volume 45 of *Series in Machine Perception and Artificial Intelligence*, pages 9–42, River Edge, NJ, USA, 2001. World Scientific Publishing Co., Inc. ISBN 981-02-4564-5.

[40] Daniel Göhring. Cooperative object localization using line-based percept communication. In U. Visser, F. Ribeiro, T. Ohashi, and F. Dellaert, editors, *RoboCup 2007: Robot Soccer World Cup XI*, Lecture Notes in Artificial Intelligence. Springer, 2008. to appear.

[41] Daniel Göhring and Hans-Dieter Burkhard. Multi robot object tracking and self localization using visual percept relations. In *Proceedings of IEEE/RSJ International Conference of Intelligent Robots and Systems (IROS), pages 31–36*. IEEE, 2006.

[42] Daniel Göhring and Jan Hoffmann. Sensor modeling using visual object relations in multi robot object tracking. In Gerhard Lakemeyer, Elizabeth Sklar, Domenico G. Sorrenti, and Tomoichi Takahashi, editors, *RoboCup 2006: Robot Soccer World Cup X*, Lecture Notes in Artificial Intelligence, pages 279–286. Springer, 2007.

[43] Daniel Göhring, Kateryna Gerasymova, and Hans-Dieter Burkhard. Constraint based world modeling for autonomous robots. In *Proceedings of Concurrency, Specification and Programming (CS&P)*, 2007. Proceedings of the CS&P 2007.

[44] Daniel Göhring, Hans-Dieter Burkhard, and Heinrich Mellmann. Constraint based object state modeling. In *European Robotics Symposium 2008, Prague, Chech Republic*, 2008. This volume (EUROS 2008).

[45] Daniel Göhring, Heinrich Mellmann, and Hans-Dieter Burkhard. Constraint based belief modeling. In *RoboCup 2008: Robot Soccer World Cup XII*, Lecture Notes in Artificial Intelligence, 2009. to appear.

[46] G. Görz, C.-R. Rollinger, and J. Schneeberger. *Handbuch der Künstlichen Intelligenz*, volume 3. Oldenbourg, 2000.

[47] Frederic Goualard and Laurent Granvilliers. Controlled propagation in continuous numerical constraint networks. *ACM Symposium on Applied Computing*, 2005.

[48] J.-S. Gutmann and D. Fox. An experimental comparison of localization methods continued. In *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2002.

[49] J.-S. Gutmann, W. Burgard, D. Fox, and K. Konolige. An experimental comparison of localization methods. In *Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 1998.

[50] Joachim Hertzberg and Frank Kirchner. Landmark-based autonomous navigation in sewerage pipes. In *In Proc. of the First Euromicro Workshop on Advanced Mobile Robots*, pages 68–73. Society Press, 1996.

[51] Jan Hoffmann and Daniel Göhring. Sensor-actuator-comparison as a basis for collision detection for a quadruped robot. In Daniele Nardi, Martin Riedmiller, Claude Sammut, and José Santos-Victor, editors, *8th International Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences)*, volume 3276 of *Lecture Notes in Artificial Intelligence*, pages 150–159. Springer, 2005.

[52] Jan Hoffmann, Matthias Jüngel, and Martin Lötzsch. A vision based system for goal-directed obstacle avoidance. In Daniele Nardi, Martin Riedmiller, Claude Sammut, and José Santos-Victor, editors, *8th International Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences)*, volume 3276 of *Lecture Notes in Artificial Intelligence*, pages 418–425. Springer, 2005.

[53] Jan Hoffmann, Michael Spranger, Daniel Göhring, and Matthias Jüngel. Making use of what you don't see: Negative information in markov localization. In *Proceedings of the 2005 IEEE/RSJ International Conference of Intelligent Robots and Systems (IROS)*. IEEE, 2006.

[54] Jan Hoffmann, Michael Spranger, Daniel Göhring, and Matthias Jüngel. Exploiting the unexpected: Negative evidence modeling and proprioceptive motion modeling for improved markov localization. In Ansgar Bredenfeld, Adam Jacoff, Itsuki Noda, and Yasutake Takahashi, editors, *RoboCup 2005: Robot Soccer World Cup IX*, Lecture Notes in Artificial Intelligence, pages 24–35. Springer, 2006.

[55] Jan Hoffmann, Michael Spranger, Daniel Göhring, Matthias Jüngel, and Hans-Dieter Burkhard. Further studies on the use of negative information in mobile robot localization. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2006.

[56] L. Jaulin and E. Walter. Set inversion via interval analysis for nonlinear bounded-error estimation. In *Automatica*, volume 29, pages 1053–1064, 1993.

[57] L. Jaulin, M. Kieffer, Didrit, and E. Walter. *Applied Interval Analysis*. Springer Verlag, London, 2001.

[58] Patric Jensfelt and Steen Kristensen. Active global localisation for a mobile robot using multiple hypothesis tracking. In *IEEE Transactions on Robotics and Automation*, pages 13–22, 1999.

[59] N. Johnson, S. Kotz, and N. Balakrishnan. *Continuous univariate distributions*, volume 1. John Wiley & Sons, New York, 1994.

[60] Simon J. Julier and Jeffrey K. Uhlmann. A new extension of the kalman filter to nonlinear systems. In *In Int. Symp. Aerospace/Defense Sensing, Simul. and Controls*, pages 182–193, 1997.

[61] Boyoon Jung and Gaurav S. Sukhatme. Cooperative tracking using mobile robots and environment-embedded, networked sensors. In *In the 2001 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 206–211, 2001.

[62] R.E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME - Journal of Basic Engineering*, 82:35–45, 1960.

[63] K. Kaplan, B. Celik, T. Mericli, C. Mericli, and L. Akin. Practical extensions to vision-based monte carlo localization methods for robot soccer domain. In Itsuki Noda, Adam Jacoff, Ansgar Bredenfeld, and Yasutake Takahashi, editors, *9th International Workshop on RoboCup 2005 (Robot World Cup Soccer Games and Conference)*, Lecture Notes in Artificial Intelligence. Springer, 2006. To appear.

[64] Rickard Karlsson and Fredrik Gustafsson. Particle filter for underwater terrain navigation. In *2003 IEEE Workshop on Statistical Signal Processing*, 2003.

[65] Michel Kieffer, Luc Jaulin, Éric Walter, and Dominique Meizel. Robust autonomous robot localization using interval analysis. *Reliable Computing*, 6(3):337–362, August 2000.

[66] Jonathan Ko and Dieter Fox. GP-BayesFilters: Bayesian Filtering Using Gaussian Process Prediction and Observation Models. In *IEEE/RSJ International Conference on Intelligent RObots and Systems (IROS)*, 2008.

[67] Daphne Koller and Raya Fratkina. Using learning for approximation in stochastic processes. In *In Proceedings of the International Conference on Machine Learning (ICML)*, pages 287–295, 1998.

[68] Benjamin Kuipers and Patrick Beeson. Bootstrap learning for place recognition. In *In Proc. 18th National Conf. on Artificial Intelligence (AAAI-2002)*, pages 174–180. AAAI/MIT Press, 2002.

[69] Cody Kwok. *Robust Real-time Perception for Mobile Robots*. PhD thesis, University of Washington, Seattle, WA, 2004.

[70] Cody Kwok and Dieter Fox. Map-based multiple model tracking of a moving object. In Daniele Nardi, Martin Riedmiller, Claude Sammut, and José Santos-Victor, editors, *8th International Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences)*, volume 3276 of *Lecture Notes in Artificial Intelligence*, pages 18–33. Springer, 2005.

[71] Martin Lauer, Sascha Lange, and Martin Riedmiller. Calculating the perfect match: An efficient and accurate approach for robot self-localization. In Ansgar Bredenfeld, Adam Jacoff, Itsuki Noda, and Yasutake Takahashi, editors, *RoboCup 2005: Robot Soccer World Cup IX*, Lecture Notes in Artificial Intelligence, pages 142–153. Springer, 2006.

[72] S. Lauritzen. *Graphical Models*. Oxford University Press, New York, 1996.

[73] Scott Lenser and Manuela M. Veloso. Sensor resetting localization for poorly modelled mobile robots. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation (ICRA 2000)*, pages 1225–1232. IEEE, 2000.

[74] Scott Lenser, James Bruce, and Manuela Veloso. CMPack: A complete software system for autonomous legged soccer robots. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 204–211. ACM Press, 2001. ISBN 1-58113-326-X.

[75] John J. Leonard and Hugh F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. In *IEEE Transactions on Robotics and Automation*, volume 7, June 1991.

[76] John J. Leonard, Hans Jacob, and S. Feder. A computationally efficient method for large-scale concurrent mapping and localization. In *Proceedings of the Ninth International Symposium on Robotics Research*, pages 169–176. Springer-Verlag, 1999.

[77] N. Lerner, Daphne Koller, Stephen Boyd, and Ronald Parr. Hybrid bayesian networks for reasoning about complex systems. Technical report, Stanford University, 2002.

[78] Li Liangqun, Ji Hongbinga, and Gao Xinboa. Maximum entropy fuzzy clustering with application to real-time target tracking. *Signal Processing*, 86:3432 – 2447, November 2006.

[79] John Maccormick and Andrew Blake. A probabilistic exclusion principle for tracking multiple objects. In *International Journal of Computer Vision*, pages 572–578, 1999.

[80] Luca Marchetti, Giorgio Grisetti, and Luca Iocchi. A comparative analysis of particle filter based localization methods. In *RoboCup 2006: Robot Soccer World Cup X*, Lecture Notes in Artificial Intelligence. Springer, 2007. to appear.

[81] Heinrich Mellmann, Matthias Jüngel, and Michael Spranger. Using reference objects to improve vision-based bearing measurements. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems IROS 2008*, pages 3939–3945, 2008. doi: 10.1109/IROS.2008.4651128.

[82] M. Montemerlo and S. Thrun. Simultaneous Localization and Mapping with Unknown Data Association Using FastSLAM. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1985–1991. IEEE, 2003.

[83] Andrew W. Moore, Jeff Schneider, and Kan Deng. Efficient locally weighted polynomial regression predictions. In *In Proceedings of the 1997 International Machine Learning Conference*, pages 236–244. Morgan Kaufmann, 1997.

[84] Ramon E. Moore and Fritz Bierbaum. *Methods and Applications of Interval Analysis (SIAM Studies in Applied and Numerical Mathematics)*. Soc for Industrial & Applied Math, 1979. ISBN 0898711614.

[85] A. Nakazawa, H. Kato, S. Hiura, and S. Inokuchi. Tracking multiple people using distributed vision systems. In *Proceedings of ICRA, pages 2974–2981*. IEEE, 2002.

[86] Viet Nguyen, Agostino Martinelli, Nicola Tomatis, and Roland Siegwart. A comparison of line extraction algorithms using 2d laser rangefinder for indoor mobile robotics. In *Proceedings of the IEEE/RSJ Intenational Conference on Intelligent Robots and Systems, IROS, Edmonton, Canada*. IEEE, 2005.

[87] Walter Nistico, Matthias Hebbel, Thorsten Kerkhof, and Christine Zarges. Cooperative visual tracking in a team of autonomous mobile robots. In Gerhard Lakemeyer, Elizabeth Sklar, Domenico G. Sorrenti, and Tomoichi Takahashi, editors, *RoboCup 2006: Robot Soccer World Cup X*, Lecture Notes in Artificial Intelligence, pages 146–157. Springer, 2007.

[88] Clark F. Olson. Probabilistic self-localization for mobile robots. *IEEE Transactions on Robotics and Automation*, 16:55–66, 2000.

[89] E. Olson, J. Leonard, and S. Teller. Fast iterative alignment of pose graphs with poor initial estimates. In *International Conference on Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE*, 2006.

[90] Stephen M. Omohundro. Bumptrees for efficient function, constraint, and classification learning. In *Advances in Neural Information Processing Systems 3*, pages 693–699. Morgan Kaufmann, 1991.

[91] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs, 2003.

[92] Michael J. Quinlan, Craig L. Murch, Richard H. Middleton, and Stephan K. Chalup. Traction monitoring for collision detection with legged robots. In Daniel

Polani, Andrea Bonarini, Brett Browning, and Kazuo Yoshida, editors, *7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conference)*, volume 3020 of *Lecture Notes in Artificial Intelligence*, pages 374–384. Springer, 2004.

[93] M. Self R. Smith and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In I. Cox and G.Wilfong, editors, *Autonomous Robot Vehicles*. Springer Verlag, 1990.

[94] Ioannis M. Rekleitis, Gregory Dudek, and Evangelos E. Milios. Multi-robot collaboration for robust exploration. *Annals of Mathematics and Artificial Intelligence*, 31(1-4):7–40, 2001.

[95] Ioannis M. Rekleitis, Gregory Dudek, and Evangelos E. Milios. Multi-robot cooperative localization: A study of trade-offs between efficiency and accuracy. In *Proc. of IEEE/RSJ International Conf. on Intelligent Robots and Systems*, pages 2690–2695, 2002.

[96] J.A. Rice. *Mathematical Statistics and Data Analysis*. Duxbury Press, second edition edition, 1995.

[97] Thomas Röfer and Matthias Jüngel. Vision-based fast and reactive monte-carlo localization. In Daniel Polani, Andrea Bonarini, Brett Browning, and Kazuo Yoshida, editors, *Proceedings of the 2003 IEEE International Conference on Robotics and Automation (ICRA)*, pages 856–861. IEEE, 2003.

[98] Thomas Röfer, Jörg Brose, Daniel Göhring, Matthias Jüngel, Tim Laue, and Max Risler. German team 2007 The German National RoboCup Team. In U. Visser, F. Ribeiro, T. Ohashi, and F. Dellaert, editors, *RoboCup 2007: Robot Soccer World Cup XI*, 2007.

[99] Raúl Rojas, Sven Behnke, Achim Liers, and Lars Knipping. FU-Fighters 2001 (global vision). In Andreas Birk 0002, Silvia Coradeschi, and Satoshi Tadokoro, editors, *RoboCup 2001: Robot Soccer World Cup V*, volume 2377 of *Lecture Notes in Artificial Intelligence*, pages 204–213. Springer, 2002.

[100] S. Roumeliotis and G. Bekey. Collective localization: A distributed kalman filter approach to localization of groups of mobile robots. In *IEEE Int. Conf. on Robotics & Automation*, pages 2958–2965, 2000.

[101] S.I. Roumeliotis and G.A. Bekey. Bayesian estimation and kalman filtering: A unified framework for mobile robot localization. In *Proc. of the IEEE International Conference on Robotics & Automation*, pages 2985 – 2992, 2000.

[102] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*, volume 2. Pearson, 2003.

[103] C. Schlieder. Representing visible locations for qualitative navigation. In N. Piera-Carrete and M. Singh, editors, *Qualitative reasoning and decision technologies*, pages 523–532. CIMNE Barcelona, 1993.

[104] T. Schmitt, R. Hanek, M. Beetz, S. Buck, and B. Radig. Cooperative probabilistic state estimation for vision-based autonomous mobile robots. *IEEE Transactions on Robotics and Automation*, 18(5):670–684, October 2002.

[105] D. Schulz, W. Burgard, D. Fox, and A. Cremers. Tracking Multiple Moving Targets with a Mobile Robot using Particle Filters and Statistical Data Association, 2001.

[106] Dirk Schulz and Dieter Fox. Bayesian color estimation for adaptive vision-based robot localization. In *Proceedings of the 2004 IEEE/RSJ International Conference of Intelligent Robots and Systems (IROS)*. IEEE, 2005.

[107] Dirk Schulz, Wolfram Burgard, Dieter Fox, and Armin B. Cremers. People tracking with mobile robots using sample-based joint probabilistic data association filters. *International Journal of Robotics Research*, 2003.

[108] E Seignez, M. Kieffer, A. Lambert, E. Walter, and T. Maurin. Experimental vehicle localization by bounded-error state estimation using interval analysis. In *Proceedings of IEEE/RSJ International Conference of Intelligent Robots and Systems (IROS), pages 1084–1089*, 2005.

[109] L.D. Stone, C.A. Barlow, and T.L Corwin. *Bayesian Multiple Target Tracking*. Mathematics in Science and Engineering. Artech House, Boston, 1999.

[110] A. Stroupe, M. Martin, and T. Balch. Distributed sensor fusion for object position estimation by multi-robot systems. In Ansgar Bredenfeld, Adam Jacoff, Itsuki Noda, and Yasutake Takahashi, editors, *Proceedings of the 2001 IEEE International Conference on Robotics and Automation (ICRA-01)*, Lecture Notes in Artificial Intelligence, pages 154–165. Springer, 2001.

[111] Ashley Stroupe and Tucker Balch. Collaborative probabilistic constraint-based landmark localization. In *Proceedings of IROS '02*, 2002.

[112] Sebastian Thrun. Particle filters in robotics. In *in Proc. 18 th Conf. Uncertainty in Artificial Intelligence*, pages 1–4, 2002.

[113] Sebastian Thrun, Dieter Fox, and Wolfram Burgard. Monte carlo localization with mixture proposal distribution. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 859–865, 2000.

[114] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA, USA, 2005.

[115] Sebatian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust monte-carlo localization for mobile robots. In *Artificial Intelligence*, volume 128(1-2), pages 99–141, 2001.

[116] Norimichi Ukita and Takashi Matsuyama. Real-time cooperative multi-target tracking by communicating active vision agents. In *Computer vision and image understanding*, volume 97, pages 137–179, 2005.

[117] Thomas Wagner and Kai Hübner. An egocentric qualitative spatial knowledge representation based on ordering information for physical robot navigation. In Daniele Nardi, Martin Riedmiller, Claude Sammut, and José Santos-Victor, editors, *8th International Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences)*, volume 3276 of *Lecture Notes in Artificial Intelligence*, pages 150–159. Springer, 2005.

[118] Eric A. Wan and Rudolph Van Der Merwe. The unscented kalman filter for non-linear estimation. In *Proceedings of Symposium 2000 on Adaptive Systems for Signal Processing, Communication and Control*, pages 153–158, 2000.

[119] Greg Welsh and Gary Bishop. An Introduction to the Kalman Filter. University of North Carolina at Chapell Hill, 2006.

List of Figures

Selbständigkeitserklärung

Ich erkläre hiermit, dass

- ich die vorliegende Dissertationsschrift “Constraint Based World Modeling for Multi Agent Systems in Dynamic Environments” selbständig und ohne unerlaubte Hilfe angefertigt habe;
- ich mich nicht bereits anderwärts um einen Doktorgrad beworben habe oder einen solchen besitze;
- mir die Promotionsordnung der Mathematisch-Naturwissenschaftlichen Fakultät II der Humboldt-Universität zu Berlin vom 17. Januar 2005 (zuletzt geändert am 13. Februar 2006 - Nr. 34/2006) bekannt ist.

Berlin, den 14. April 2009

Daniel Göhring